ALGORYTMY

ZESZYT SPECJALNY

INSTYTUT MASZYN MATEMATYCZNYCH

A L G O R Y T M Y

Zeszyt Specjalny . 1974

MATERIAŁY z SYMPOZJUM
"METODY PROGRAMOWANIA I OPISU
SYSTEMÓW OPERACYJNYCH"

Warszawa, 18-20.X.1973

PL ISSN 0065-6240

Opracowanie merytoryczne zeszytu:

ZESPÓŁ REDAKCYJNY w składzie
Tadeusz Englert
Jerzy Mysior
Jacek Olszewski
Andrzej Rowicki

Redaktor techniczny: Maria KOZŁOWSKA

Adres Redakcji: Warszawa, ul. Krzywickiego 34, tel. 28-37-29

Druk wykonano z materiałów opracowanych i przepisanych na kredzie przez Zleceniodawcę

WPM "WEMA". Warszawa 1974 r. Wyd. I. Nakład 550+2 egz. Ark.wyd. 10,62. Ark. druk. 18,0. Zam. 1620/74-6-Z/S 0-3 [redacted].

Druk: WPM "WEMA" Zakład Poligraficzny w B-stoku Zam. 335/74.

OD REDAKCJI

Niniejszy specjalny numer ALGORYTMÓW stanowi zbiór referatów wygłoszonych na Sympozjum: "Metody programowania i opisu systemów operacyjnych" w Warszawie w dniach 18-20 października 1973, zorganizowanym przez Zakład Teorii Systemów Operacyjnych

Zespół Redakcyjny, pełniący rolę zarówno recenzentów, jak i redaktorów tego numeru, postawił sobie za cel przekazać do druku dostarczone przez referentów teksty w postaci możliwie jednolitej tematycznie i terminologicznie. Członkowie Zespołu zdają sobie sprawę, że mimo poczynionych zmian i skreśleń, nie udało się w pełni osiągnąć celu. Powodów tego stanu rzeczy łatwo się domyślić, po pierwsze, tematem sympozjum była dziedzina, w której panuje jeszcze chaos pojęciowy, po drugie, sytuację pogarsza jeszcze brak odpowiedniej terminologii polskiej, w której występują często terminy angielskie, jednakże różnie przez różnych autorów rozumiane i w różnych znaczeniach używane. Dlatego też Czytelnik znajdzie w tekście numeru odnośniki pochodzące od Zespołu Redakcyjnego, które, mamy nadzieję, będą pomocne w czytaniu.

Wszystkie zmiany i skreślenia, o których mowa wyżej, są natury wyłącznie terminologicznej lub dotyczą zakresu tematycznego Sympozjum. Zespół nie wprowadził żadnych zmian, ani nie skreślił zdań, wyrażających poglądy odmienne od poglądów członków Zespołu. Czytelnik może więc znaleźć w niniejszym wydawnictwie różne poglądy i sposoby potraktowania tych samych problemów, różne podejścia i sposoby rozumowania prowadzące do stworzenia systemu operacyjnego i jego dokumentacji.

Zbiór materiałów otwiera referat wstępny W.M. Turskiego, stanowiący uzasadnienie traktowania problemów konstruowania systemów operacyjnych jako problemów naukowych, pilnie wymagających uporządkowania zbioru pojęć tej dziedziny i stworzenia odpowiednich metod, które umożliwią szybsze konstruowanie systemów operacyjnych, możliwie poprawnie działających.

Układ pozostałych materiałów Sympozjum tylko z grubsza pokrywa się z jego programem. Program był sporządzony z uwzględnieniem ram czasowych poszczególnych sesji. Natomiast w druku staramy się zachować jedność tematyczną poszczególnych części, na jakie cały materiał został podzielony.

W części dotyczącej synchronizacji procesów Czytelnik znajdzie referat przeglądowy T. Englerta, traktujący o sposobach zapisu synchronizacji równoległych procesów i o próbach wykazywania tego, że synchronizacja jest poprawna. Temu tematowi również jest poświęcony komunikat S. Chrobota, opisujący pewien minimalny zbiór operacji synchronizacyjnych, wystarczający do zaprogramowania dowolnie złożonych struktur procesów równoległych.

Odrębną część materiałów stanowią prace A.C. Rowickiego dotyczące sposobów wyliczenia, ile trzeba procesorów do zrealizowania równoległych procesów o zadanej strukturze oraz wyliczenia minimum czasu, w którym procesy te zostaną zrealizowane przy zadanej liczbie procesorów. Pierwsza z nich jest wprowadzeniem do zagadnienia, potraktowanym przeglądowo, druga - komunikatem podającym sposób wyliczenia tych wielkości dla grafowych struktur procesów równoległych.

W części dotyczącej modelowania systemów cyfrowych mamy referaty przeglądowe P. Bielkowicza i P. Perkowskiego oraz J. Marońskiego. Pierwszy z nich omawia metody modelowania i symulacji takich systemów od strony pojęć i reguł przy tym stosowanych. Drugi zawiera również przegląd zrealizowanych i stosowanych systemów modelowania i symulacji zarówno programowych jak i sprzętowych.

Następna część materiałów zawiera referaty traktujące o problemach i metodach programowania oraz dokumentowania systemów operacyjnych. Pierwszy z nich H. i J. Mysiorów jest próbą zrealizowania tego, co się określa jako warstwową i modularną budowę systemów operacyjnych. Drugi, J. Olszewskiego, traktuje o możliwościach strukturalnego programowania systemów operacyjnych w językach tzw. wyższego poziomu. Następny referat J. Pieli i W. Skurzaka jest krótkim komunikatem o sposobie rozszerzenia możliwości funkcjonalnych programu Egzekutor dla maszyny ODRA 1304. Komunikat Z. Kosowskiego i Z. Bytnerowicza dotyczy sposobu konstruowania zestawu programów manipulacyjnych jako część dużego systemu operacyjnego. Ostatni z referatów tej części jest również krótkim komunikatem na temat dokumentacji systemów operacyjnych potraktowanej niejako oddzielnie od samego programowania.

Kolejną i ostatnią już część materiałów stanowią komunikaty o niektórych właśnie zrealizowanych lub nawet jeszcze realizowanych systemach operacyjnych i o sposobach ich realizacji. Całą tę część można potraktować jako zbiór ilustracji niektórych ogólnych tez wypowiedzianych w referatach przeglądowych i problemowych umieszczonych w poprzednich częściach materiałów. Zestawiając ze sobą referaty ogólne i ten zbiór przykładów Czytelnik może przekonać się, że w większości przypadków ilustrują one trudności systematycznego i strukturalnego podejścia do problemu konstruowania systemów operacyjnych.

TREŚĆ

ALGORYTMY, Zeszyt Specjalny, 1974, 9-18

681.322.06:
061.3/438/"1973"

ZAGAJENIE SYMPOZJUM

Władysław M. TURSKI

Instytut Maszyn Matematycznych
Warszawa, ul. Krzywickiego 34

Pracę złożono 10.I.1974

Wymieniono praktyczne powody, dla których należy prowadzić badania w zakresie systemów operacyjnych. Należą do nich: konieczność budowy wielu systemów operacyjnych dla różnorodnych zastosowań systemów minikomputerów oraz dla niestandardowych zastosowań dużych maszyn. Podano kilka przykładów problemów naukowych występujących przy badaniach systemów operacyjnych, a mianowicie: współdziałanie niezależnych procesów, styk pomiędzy sprzętem i jego oprogramowaniem, przejmowanie funkcji oprogramowania przez sprzęt, nowe problemy optymalizacji. Wymieniono i pobieżnie przeanalizowano nierozwiązane problemy w dziedzinie systemów operacyjnych, takie jak: metody projektowania, poprawność, optymalizacja, współdziałanie i styk pomiędzy sprzętem i oprogramowaniem oraz pomiędzy systemami operacyjnymi i translatorami.

Zanim przejdę do treści merytorycznej mojego referatu, chciałbym przywitać uczestników Sympozjum w imieniu Dyrekcji Instytutu Maszyn Matematycznych oraz w imieniu naszego macierzystego Zjednoczenia "MERA", z radością witającego seminaria i sympozja naukowe organizowane z tak szeroką reprezentacją pracowników nauki i praktyków informatyki. Jest mi szczególnie przyjemnie powitać uczestników sympozjum z zakresu teorii systemów operacyjnych, zwłaszcza, że jest to dziedzina, którą bodajże 7 lat temu nikt jeszcze się w Polsce nie zajmował, a teraz znalazła się wystarczająca liczba referatów i komunikatów, ażeby zorganizować 2 sympozja w odstępie zaledwie dwutygodniowym (następne sympozjum organizuje Zakład Systemów Automatyzacji Kompleksowej PAN w dniach od 5 do 10 listopada).

W ciągu 3-dniowych obrad wysłuchamy kilkunastu referatów i komunikatów, które będą miały poważną treść techniczną, tzn. będą zawierały informacje liczbowe, wzory, wykresy itp. Jest to oczywiście oznaką znacznego rozwoju tej dziedziny; zaczynają się formować szkoły, teorie. W niedługim czasie zaczniemy tworzyć systemy operacyjne i modele systemów operacyjnych oparte na ścisłych kanonach; system operacyjny to będzie piątka czy szóstka, oczywiście uporządkowana, która będzie miała ściśle określone własności i jeżeli systemy operacyjne w rzeczywistości własności tych posiadać nie będą, to tym gorzej dla rzeczywistości. Systemem operacyjnym będzie nie to, co jest na maszynie, ale ta piątka, szóstka, czy ósemka, a może kiedyś dwunastka i trzynastka uporządkowana, jako że system operacyjny jest bardzo skomplikowanym układem. Jeszcze trochę, a dojdziemy w tej dziedzinie do wymaganych kryteriów "akademickości" dyscypliny.

Skoro więc oczekuje nas spora porcja komunikatów i referatów technicznych, ścisłych, ja zajmę trochę czasu rozważaniami natury luźniejszej. Wydaje mi się, że czasami warto na jakąś dziedzinę ludzkich umiejętności, żeby nie powiedzieć wiedzy, popatrzeć nie tyle z perspektywy publikacji, z perspektywy bardzo rygorystycznego wzoru, czy z jeszcze bardziej rygorystycznego układu aksjomatów, i nawet nie z punktu widzenia konkretnej realizacji, na konkretnej maszynie, w konkretnym czasie, dla konkretnych użytkowników, ale z punktu widzenia badacza, który przecież podejmuje trud zajmowania się jakąś dyscypliną również i dlatego, że interesują go pytania: dlaczego, po co, gdzie w układzie rzeczy to, czym się zajmuje jest umiejscowione i jak się wiąże ze swym otoczeniem. Takimi rozważaniami chciałbym się z państwem teraz podzielić.

Pierwszą sprawą będzie pytanie - dlaczego zagadnienia systemów operacyjnych są ważne?

Patrząc na to co się dzieje w zakresie instalacji maszyn, widzi się te wielkie, bardzo drogie maszyny instalowane, dostarczane z gotowym systemem operacyjnym, lepszym czy gorszym, ale prawie zawsze zbyt wielkim, żeby działaniem jednostkowym

czy nawet działaniem niewielkiej grupy można było zmienić, ulepszyć, zaadaptować taki system operacyjny. Innymi słowy, kiedy mamy do czynienia z behemotami typu systemów operacyjnych IBM, właściwie stajemy się bezradni, to już nie jest obiekt, który można zmieniać. To jest coś jak masa Ziemi. Nie poddaje się zmianom. To jest constant. Nawet biorąc pod uwagę to, że funkcją programów jest u nas w kraju głównie pewna adaptacja, docieranie, dopasowywanie istniejących systemów operacyjnych do jednego, dwóch, może pięciu urządzeń, to koncepcyjnie rzecz jest stała.

Jednakże, równocześnie z pojawieniem się tych behemotów, obserwujemy inwazję, jeśli można tak powiedzieć, minikomputerów różnej proweniencji, które nie są z natury rzeczy przeznaczone do eksploatacji w sposób standardowy. A że mają one zadziwiającą zdolność do przejawiania się w bardzo różnorodnych konfiguracjach, jest chyba praktyczną potrzebą umieć robić systemy operacyjne dla systemów, w których minikomputery pojedynczo, albo w zespołach odgrywają rolę procesorów. Relatywna taniość tych maszyn powoduje, że modele ich użytkowania będą bardzo różne. W przeciwieństwie do maszyn wielkich, w których model użytkowania jest jeden, dwa, być może trzy i gdzie wobec tego są jeden, dwa czy trzy warianty systemu operacyjnego, używane faktycznie w wielu tysiącach egzemplarzy każdy; systemy operacyjne dla licznych konfiguracji i zastosowań przeróżnych systemików czy systemów minikomputerowych będą miały zapewne dużo większą różnorodność. A więc powstaje praktyczna potrzeba zdobycia umiejętności robienia systemów operacyjnych.

Druga potrzeba praktyczna robienia systemów operacyjnych, występująca znacznie rzadziej, ale chyba bardziej atrakcyjna – to niestandardowe systemy z wykorzystaniem maszyn dużych. Liczbowo jest ich chyba niewiele, tym niemniej będą występowały i będą zapewne wymagały unikalnych systemów operacyjnych dostosowanych do takich niestandardowych zastosowań. Przykładem niestandardowych zastosowań jest chociażby system rezerwacji biletów, bo w końcu będzie w Polsce jeden system rezerwacji biletów kolejowych. Być może będzie drugi system rezerwacji biletów lot-

niczych, aczkolwiek na razie patrząc na rozkład lotów wewnętrznych linii PLL "Lotu", wydaje się, że taki system można by znakomicie zorganizować bez jakiegokolwiek komputera. Ponieważ systemów rezerwacji miejsc nie będzie dużo, więc również nie będzie zapewne standardowego systemu rezerwacji miejsc - będzie to tylko jeden system rezerwacji miejsc z własnym, unikalnym systemem operacyjnym.

Trzecia kategoria praktycznych potrzeb konstruowania systemów operacyjnych wyniknie z wysoce niestandardowych konfiguracji sprzętowych. Nie mam na myśli minikomputerów, bo w ich przypadku zakładam, że normą jest fakt niestandardowości konfiguracji, zwłaszcza w zastosowaniach w sterowaniu i w eksperymentach, ale również duże maszyny mogą występować w bardzo niestandardowych konfiguracjach, np. są to maszyny cyfrowe użyte w roli central telekomunikacyjnych, gdzie procesor właściwie bardzo mało przetwarza, a bardzo dużo przełącza. Takie niestandardowe konfiguracje i zastosowania będą wymagały unikalnych systemów operacyjnych, a więc wzbudzą konkretne potrzeby wykonawcze i postawią problem umiejętności efektywnego wykonywania systemów operacyjnych. Jeśli jednak ma to być tak, to nie wystarcza uczyć się o systemach operacyjnych, żeby umieć robić systemy operacyjne trzeba pewnej praktyki.

Drugim powodem dlaczego zagadnienia systemów operacyjnych są ważne, a może nawet ważniejszym są względy poznawcze. Najpierw chciałbym trochę zareklamować systemy operacyjne. Co prawda, w gronie uczestników tego sympozjum nie bardzo ma to sens, ale chyba warto zdać sobie sprawę, że właśnie przy studiowaniu zagadnień systemów operacyjnych po raz pierwszy zetknęliśmy się z rzeczywistymi, konkretnymi zagadnieniami, w których jednoczesność działania i problemy synchronizacji są sprawą podstawową. Dopóki nie weszliśmy w zagadnienia systemów operacyjnych, świadomie lub nieświadomie (a nastąpiło to zapewne wcześniej nim powiedzieliśmy wyraźnie, że zaczęliśmy zajmować się systemami operacyjnymi), w badaniach naukowych można było stosować zasadę divide et impera. Co więcej, jest to klasycznym zaleceniem badawczym - "rozłóż problem na części składowe i studiuj każdą z nich oddzielnie". Nie chcę przez to powiedzieć, że systemów

operacyjnych nie chcielibyśmy też tak badać - właśnie tak chcielibyśmy je badać. Tyle tylko, że podziału tego nie można przeprowadzić tak łatwo i prosto jak w systemach, w których jednoczesność działania i problemy współdziałania, synchronizacji, problemy interferencji pożytecznej i interferencji szkodliwej procesów współbieżnych nie występują. Otóż z taką ostrością problemy te wystąpiły po raz pierwszy dopiero przy systemach operacyjnych. Oczywiście, wszystko to zaczyna się teraz przeradzać w osobną gałąź badań. Zawsze bowiem tak jest, że problem zrodzony w warunkach kontaktu ze światem rzeczywistym, po pewnym czasie degeneruje się, czy wyradza w przedmiot "czystych" zainteresowań naukowych, nie czerpiących już materiału z konfrontacji z rzeczywistością, bo mających dość materiału założonego i wygenerowanego przez sprawnie działający aparat rozbudowy aksjomatów i twierdzeń.

Drugi, w moim przekonaniu równie atrakcyjny, aspekt poznawczy systemów operacyjnych polega na tym, że właśnie poprzez systemy operacyjne, przez ich studiowanie uzyskuje się właściwy poziom abstrakcji dla analizy architektury systemu liczącego. Analizowanie, czy zastanawianie się nad problemami architektury systemów liczących, przynajmniej mnie osobiście, najłatwiej prowadzić z punktu widzenia systemu operacyjnego. Właśnie przez system operacyjny najdogodniej widzę zarówno konfigurację, jak i architekturę. Na poziomie systemu operacyjnego jestem dostatecznie uwolniony od szczegółów technicznych, tzn. widzę je z odpowiedniej perspektywy, a jednocześnie dostrzegam wszystkie funkcje sprzętowe, które są interesujące z punktu widzenia eksploatacji (a nie konserwacji).

Co więcej, właśnie badając systemy operacyjne, i z ich punktu widzenia architekturę systemów, dochodzi się do pewnych uwag odnośnie zmian w konstrukcjach sprzętowych. Niekiedy mówi się o tym, że jest to przejmowanie funkcji oprogramowania przez sprzęt. Oczywiście jest to tylko kwestia akcentów słownych, i wydaje mi się, że nie jest to tylko przejmowanie funkcji oprogramowania przez sprzęt, lecz także przejmowanie przez projektantów sprzętu wyników badań systemów operacyjnych.

ALGORYTMY, Zeszyt Specjalny, 1974, 9-18

681.322.06:
061.3/438/"1973"

ZAGAJENIE SYMPOZJUM

Władysław M. TURSKI

Instytut Maszyn Matematycznych
Warszawa, ul. Krzywickiego 34

Pracę złożono 10.I.1974

Wymieniono praktyczne powody, dla których należy prowadzić badania w zakresie systemów operacyjnych. Należą do nich: konieczność budowy wielu systemów operacyjnych dla różnorodnych zastosowań systemów minikomputerów oraz dla niestandardowych zastosowań dużych maszyn. Podano kilka przykładów problemów naukowych występujących przy badaniach systemów operacyjnych, a mianowicie: współdziałanie niezależnych procesów, styk pomiędzy sprzętem i jego oprogramowaniem, przejmowanie funkcji oprogramowania przez sprzęt, nowe problemy optymalizacji. Wymieniono i pobieżnie przeanalizowano nierozwiązane problemy w dziedzinie systemów operacyjnych, takie jak: metody projektowania, poprawność, optymalizacja, współdziałanie i styk pomiędzy sprzętem i oprogramowaniem oraz pomiędzy systemami operacyjnymi i translatorami.

Zanim przejdę do treści merytorycznej mojego referatu, chciałbym przywitać uczestników Sympozjum w imieniu Dyrekcji Instytutu Maszyn Matematycznych oraz w imieniu naszego macierzystego Zjednoczenia "MERA", z radością witającego seminaria i sympozja naukowe organizowane z tak szeroką reprezentacją pracowników nauki i praktyków informatyki. Jest mi szczególnie przyjemnie powitać uczestników sympozjum z zakresu teorii systemów operacyjnych, zwłaszcza, że jest to dziedzina, którą bodajże 7 lat temu nikt jeszcze się w Polsce nie zajmował, a teraz znalazła się wystarczająca liczba referatów i komunikatów, ażeby zorganizować 2 sympozja w odstępie zaledwie dwutygodniowym (następne sympozjum organizuje Zakład Systemów Automatyzacji Kompleksowej PAN w dniach od 5 do 10 listopada).

prac i sporo metod dowodzenia poprawności programów, co od razu powoduje, że trzeba określić "poprawność programu". Ja mówię ostrożniej, chodzi mi o dowodzenie poprawności sądów orzekanych o programach. Tutaj spodziewamy się bardzo wiele po badaniach teoretycznych, jak zwykle przy sprawach, które nas silnie pasjonują. Ale droga ta okazuje się pełna cierni i powolna, bo nawet dla programu sortowania wewnętrznego dowód nie jest zbyt prosty. Zdaje się, że dla bardziej złożonych programów, żadnych dowodów poprawności dotychczas nie podano. A przecież, mimo że program sortowania nie jest problemem trywialnym, daleko mu do złożoności zagadnienia systemu operacyjnego. Może więc nie stawiać sobie zadania dowodzenia orzeczeń o systemach operacyjnych, może zastanowić się nad metodami weryfikacji tych systemów, bo z tym też jest niedobrze. Systemy operacyjne sprawdzamy na tak nieznacznej liczbie przypadków, że gdyby w pięcio- czy sześcioletniej eksploatacji tyle tylko było pomyłek, ile sukcesów w czasie testowania, to taki system byłby niezły. Oczywiście zagadnienie weryfikacji łączy się z problemem projektowania systemów - dążenie do weryfikowalności systemu operacyjnego powinno być równie ważnym kryterium projektowym jak funkcjonalna niezawodność, innymi słowy być może dążenie do weryfikowalności nie jest najważniejszym kryterium projektowania systemu operacyjnego, ale powinno być jednym z najważniejszych.

Trzecim problemem badawczym są metody adjustacji systemów operacyjnych. Może niezręcznie użyłem słowa "adjustacja", ale chodzi mi tutaj o słowo ogólniejsze niż te trzy podproblemy, które chcę wyliczyć, a więc: zagadnienie przystosowalności, przez co rozumiem zmiany konstrukcji systemu operacyjnego pozwalające dostosować go do potrzeb użytkownika, zagadnienie rozszerzalności czyli dostosowanie systemu operacyjnego do zmian konfiguracji i zagadnienie przenośności, tj. przenoszenie systemu operacyjnego z konfiguracji na konfigurację, przy istotnych różnicach między nimi. O ile rozszerzalność dotyczy rozszerzenia konfiguracji w obrębie rodziny, czyli np. dodania pamięci czy urządzeń, to przenośność oznacza przenoszenie systemu operacyjnego z maszyny na inną maszynę. Należałoby mówić o ewentualnej przenośności, bo czy system operacyjny może być

przenośny, czy powinien być przenośny, są to właśnie zagadnienia badawcze, których rozwiązań jeszcze nie znamy. Otóż metody adjustacji, obejmujące tę trójkę problemów, stanowią bardzo ważny problem badawczy wiążący się z metodami projektowania i automatyzacji procesów projektowania, z aspektem lingwistycznym systemów operacyjnych.

Na czwartym miejscu wyliczymy problemy optymalizacji i badania jakości. Weryfikacja, czy jeszcze lepiej dowodzenie poprawności orzeczeń o systemach operacyjnych w zasadzie są metodami binarnymi. System spełnia wymagania albo ich nie spełnia. Teoretycznie rzecz biorąc, można sobie pomyśleć, że w jednym celu konstruujemy kilka systemów operacyjnych. Wtedy powstaje problem, który z nich jest lepszy. Na to pytanie w tej chwili nie umiem rzetelnie odpowiedzieć.

Piąty problem, który wydaje mi się godny uwagi badaczy, to zagadnienie współdziałania oprogramowania i sprzętu, i jak już mówiłem, zagadnienie to rozpatruję zwykle z punktu widzenia systemu operacyjnego. Jako przykład chciałbym podać następujący problem badawczy: czy nie jest za dużym ograniczeniem to, że każde przerwanie maskuje wszystkie inne przerwania, czy to nie wyniknęło z pewnego prymitywnego poglądu na budowę oprogramowania, jaki panował 12-15 lat temu, kiedy to dopiero konstruowano pierwsze systemy przerwań. Innymi słowy, problem polega na zbadaniu jaką informację naprawdę trzeba przechowywać w momencie różnych przerwań i jakie kombinacje przerwań powinny być wzajemnie maskowane, bo napewno nie wszystkie. Z punktu widzenia systemu operacyjnego jest to problem ciekawy, badawczy. A takich problemów, jak się dobrze przyjrzeć, jest bardzo dużo.

I wreszcie szósty problem, to styk systemu operacyjnego z translatorami. Nie przebadano tego zagadnienia nawet w systemach tradycyjnych, niekonwersacyjnych, nie mówiąc już o systemach konwersacyjnych. Problem ten nosi również nazwę problemu wiązania parametrów. W każdym systemie i translatorze jest pewna liczba parametrów, które wymagają ustaleń, np. parametry rozmieszczenia, adresacji itp. Zagadnienie polega na okreś-

leniu, kiedy te parametry należy ustalić. Nie wiemy tego nawet w systemach tradycyjnych, systemach wsadowych, systemach z podziałem czasu, a w systemach konwersacyjnych, gdzie wszystko może się zmieniać dynamicznie, ten styk translatora z systemem operacyjnym wymaga gruntownego przebadania.

Na zakończenie, spróbuję przepowiedzieć jakie będą nowe kierunki badań systemów operacyjnych w najbliższych latach. Po pierwsze, systemy operacyjne w krotnych systemach liczących, przy czym krotność nie będzie rzędu kilku, kilkunastu czy kilkudziesięciu, ale kilkuset czy kilku tysięcy, np. system operacyjny dla systemu cyfrowego składającego się z kilku tysięcy procesorów. Nie mam tu na myśli procesorów dużych, wielkich maszyn, mam na myśli sieci liczące. Wiąże się to bezpośrednio z metodologią programowania dla systemów o takim profilu.

Po drugie, systemy operacyjne maszyn bezpamięciowych, mam na myśli maszyny bez pamięci operacyjnej, maszyny stanowiące jedną wielką macierz licząco-pamiętającą.

I wreszcie ostatni problem, który wydaje mi się ważny na lata najbliższe - to systemy operacyjne dynamicznie, konwersacyjnie adjustowane.

ВВОДНОЕ СЛОВО

Резюме

Перечисление причин, по которым следует заниматься операционными системами, а именно: необходимость разработки многих операционных систем для различных применений малых вычислительных машин и для нестандартных применений больших вычислительных систем. Примеры общенаучных проблем, которые встречаются при исследовании операционных систем, в том числе: проблемы синхронизации независимых процессов, связи между конструкцией машины и ее матобеспечением, перенятие машиной функций программирования, новые задачи оптимизации. Перечисление и краткий анализ нерешенных проблем из области операционных систем: методики проектирования, проверки корректности, оптимизации, методов приспосабливания, методов улучшения качества, связи между конструкцией машины и ее матобеспечением, взаимодействия и стыковки операционных систем и трансляторов.

INTRODUCTION TO SYMPOSIUM

Summary

Practical reasons for research in operating systems are listed: the necessity to construct many operating systems for diversified applications of minicomputer systems and for nonstandard applications of computers. Some examples of general scientific problems arising in operating systems investigations are given: cooperation of independent processes, interface between hardware and software, swapping of software functions for hardware realizations, new kinds of optimization problems. Following unsolved problems in the area of operating systems are listed and briefly analysed: design methods, correctness, adaptation methods, optimization, hardware-software replacement and interface, interface between operating systems and compilers.

I. SYNCHRONIZACJA PROCESÓW

ALGORYTMY, Zeszyt Specjalny, 1974, 21-51

681.322.06:
061.3/438/"1973"

METODY OPISU SYNCHRONIZACJI PROCESÓW

Tadeusz ENGLERT

Instytut Maszyn Matematycznych
Warszawa, ul. Krzywickiego 34

Pracę złożono 10.I.1974

W pracy dokonano przeglądu metod rozwiązywania problemu synchronizacji.

- Programowe rozwiązanie Dijkstry, w którym użyto jedynie zmiennych logicznych i całkowitych oraz nie założono istnienia specjalnych instrukcji języka wysokiego poziomu.
- Formalizm Gilberta i Chandlera umożliwiający automatyczną weryfikację rozwiązania.
- Propozycja Lampsona, który definiuje specjalne instrukcje maszynowe TSL i PRO ułatwiające synchronizację procesów współbieżnych.
- Rozwiązanie problemu synchronizacji przy użyciu semaforów i operacji P i V Dijkstry.
- Formalizm Habermanna, w którym można dowodzić własności rozwiązania danej synchronizacji.

S p i s   t r e ś c i

1. WSTĘP

Ciągle rosnące zapotrzebowanie na moc obliczeniową powoduje, że buduje się coraz większe i coraz bardziej skomplikowane systemy liczące. Wzrost złożoności współczesnych systemów zwiększa stopień skomplikowania ich systemów operacyjnych. Dotyczy to zwłaszcza systemów wieloprocesorowych, umożliwiających jednoczesne wykonywanie wielu programów. W systemach tych występuje ostro problem synchronizacji procesów współbieżnych, stanowiący jeden z czynników oddziaływujących na złożoność systemu operacyjnego.

Można rozróżnić dwa sposoby współdziałania procesów:

- wykluczanie
- współpraca przez bufor.

Wykluczanie

Procesy mają dostęp do wspólnego zbioru zmiennych. Problem wykluczania polega na takim zsynchronizowaniu procesów, aby w danej chwili tylko jeden proces działał na zmiennych ze wspólnego zbioru. Ten fragment procesu, który realizuje komunikację ze wspólnym zbiorem zmiennych przyjęto nazywać rejonem krytycznym.

Współpraca przez bufor

Są dwie klasy procesów. Procesy zwane nadajnikami, wytwarzające komunikaty oraz procesy zwane odbiornikami, przetwarzające komunikaty. Nadajnik po wytworzeniu komunikatu umieszcza go w buforze; odbiornik pobiera komunikat z buforu, a następnie przetwarza go. Pojemność buforu zależy od tego, jak dalece nadajniki mają wyprzedzać w działaniu odbiorniki.

Przy umieszczaniu komunikatu w buforze nadajnik i odbiornik powinny być zsynchronizowane ze względu na przepełnienie buforu. Umieszczenie komunikatu musi być opóźnione, gdy w buforze nie ma wolnego miejsca; opóźnienie powinno ulec likwidacji po pojawieniu się wolnego miejsca.

Przy pobieraniu komunikatu nadajnik i odbiornik powinny być zsynchronizowane ze względu na pusty bufor. Pobranie komunikatu musi być opóźnione, jeśli nie ma w buforze nieprzetworzonych komunikatów; opóźnienie powinno być usunięte po nadejściu pierwszego komunikatu.

W pracy [3] Dijkstra przeprowadził analizę i podał rozwiązanie problemu wykluczania stosując wyłącznie środki programowe. Podejście Dijkstry omówiono w rozdziale 2.

Programowe rozwiązanie problemu wykluczania ma dwie wady:

1. rozwiązanie już dla dwóch procesów jest skomplikowane. Utrudniona jest w związku z tym analiza własności rozwiązania,

2. występuje tzw. problem "aktywnego oczekiwania", co powoduje zmniejszenie efektywności funkcjonowania systemu procesów jako całości.

Gilbert i Chandler [5] zastosowali aparat teorii automatów skończonych umożliwiający formalną analizę własności rozwiązań problemu wykluczania. Formalizm ten omówiono w rozdziale 4. Drugą z wyżej wymienionych wad eliminuje propozycja Lampsona [11], którą omówiono w rozdziale 3. Lampson zakłada istnienie specjalnego rozkazu maszyny - rozkazu TSL - ułatwiającego synchronizację w odróżnieniu od wyżej wspomnianego rozwiązania Dijkstry, w którym synchronizację realizuje się za pomocą instrukcji występujących w każdej maszynie, a mianowicie instrukcji nadawania wartości zmiennej i testowania zmiennej.

Do grupy rozwiązań problemu synchronizacji, w których postuluje się istnienie specjalnych operacji, można zaliczyć metodę synchronizacji za pomocą operacji P i V wykonywanych na specjalnych zmiennych zwanych semaforami, zaproponowaną przez Dijkstrę [3]. Metodę tę omówiono w rozdziale 3.

W rozdziale 5 omówiono propozycję Habermanna [7], który podał formalną definicję operacji P i V oraz udowodnił twierdze-

nie umożliwiające dowodzenie, że dane rozwiązanie synchronizacji ma określoną własność.

Rozdział 6 zawiera krótkie omówienie sieci Petri.

2. PROGRAMOWE ROZWIĄZANIE PROBLEMU WYKLUCZANIA

Rozważmy dwa procesy cykliczne współbieżne, które w każdym cyklu korzystają ze wspólnego zbioru zmiennych. Aby współpraca między procesami była poprawna musi być spełniony

Warunek 1. W każdej chwili najwyżej jeden proces może się znajdować w swym rejonie krytycznym. Rozwiązanie podamy w zmodyfikowanym ALGOL-u, a mianowicie zapis

So parbegin S1, S2,... Sn parend Sn+1

oznacza, że po wykonaniu instrukcji So następuje jednoczesne wykonywanie instrukcji S1, S2,..., Sn. Instrukcja Sn+1 zostanie wykonana po wykonaniu wszystkich instrukcji S1, S2,..., Sn.

Wersja 1

```
begin integer turn; turn:=1;
   parbegin
      process 1: begin L1: if turn=2 then goto L1;
                 critical section 1;
                 turn:=2;
                 remainder of cycle 1; goto L1
                 end
      process 2: begin L2: if turn=1 then goto L2;
                 critical section 2;
                 turn:=1;
                 remainder of cycle 2; goto L2
                 end
      parend
   end
```

Gdy zmienna turn=1 to oznacza, że proces 1 wszedł do swego rejonu krytycznego.

Ponieważ w każdej chwili turn=1 lub 2 więc dla wersji 1 warunek 1 jest spełniony. Jednakże rozwiązanie to ma poważny mankament, ponieważ narzuca cykl

proces 1, proces 2, proces 1,......

Gdy jeden z procesów zatrzyma się w części "remainder of cycle" (tylko taki stop jest dozwolony) powoduje to zatrzymanie się drugiego procesu. Dlatego powinien być spełniony

Warunek 2. Zatrzymanie się procesu na zewnątrz rejonu krytycznego nie powinno mieć wpływu na działanie innych procesów.

Wersja 2

```
begin integer c1, c2;
    c1:=1; c2:=1
  parbegin
    process 1: begin A1: c1:=0;
                     L1: if c2=0 then goto L1;
                         critical section 1;
                         c1:=1;
                         remainder of cycle 1; goto A1
                     end
    process 2: begin A2: c2:=0;
                     L2: if c1=0 then goto L2;
                         critical section 2;
                         c2:=1;
                         remainder of cycle 2; goto A2
                     end
  parend
end
```

Łatwo sprawdzić, że wersja 2 spełnia warunek 1 i 2. Jest to jednak rozwiązanie nie do przyjęcia, ze względu na następującą możliwość. Przypuśćmy, że wykonały się jednocześnie instrukcje c1:=0, c2:=0, wówczas żaden z procesów nie wejdzie do swego rejonu krytycznego. Dlatego wymaga się aby był spełniony

Warunek 3. Decyzje odnośnie tego, który z procesów ma wejść do swego rejonu krytycznego musi być podjęta w skończonym

czasie. Poprawne - tzn. takie, które spełnia warunki 1, 2 i 3 - rozwiązanie problemu wykluczania dla dwóch procesów podał Th. J. Dekker (Dijkstra [3])

```
begin integer c1, c2, turn;
    c1:=1; c2:=1; turn:=1;
    parbegin
    process 1: begin A1: c1:=0;
                     L1: if c2=0 then
                         begin if turn=1 then goto L1;
                         c1:=1;
                     B1: if turn=2 then goto B1;
                         goto A1
                     end
                         critical section 1;
                         turn:=2; c1:=1;
                         remainder of cycle 1; goto A1
               end
    process 2: begin A2: c2:=0;
                     L2: if c1=0 then
                         begin if turn=2 then goto L2;
                         c2:=1;
                     B2: if turn=1 then goto B2;
                         goto A2
                     end
                         critical section 2;
                         turn:=1; c2:=1;
                         remainder of cycle 2; goto A2
                     end
           parend
        end
```

Rozwiązanie to jest dość złożone i udowodnienie, że jest ono poprawne nie jest sprawą prostą.

Sprawa jeszcze bardziej się komplikuje w wypadku współdziałania n procesów. Problem ten rozwiązał Dijkstra w [2]. Jak sam wyznał w [4] był to najtrudniejszy program jaki kiedykolwiek napisał. Po opublikowaniu pracy [2] Knuth [10] zauważył,

że w przypadku n, (n >2) procesów warunek 3 jest niewystarczający. Może się zdarzyć, że mimo spełnienia warunku 3 proces może nigdy nie wejść do swego rejonu krytycznego. Knuth zmodyfikował warunek 3 następująco

Warunek 3'. Każdy proces, który zgłasza gotowość wejścia do swego rejonu krytycznego, musi do niego wejść w skończonym czasie. Rozwiązanie w ten sposób zmodyfikowanego problemu wykluczania dla n procesów podał Knuth w [10].

3. SEMAFORY

Przyczyną złożoności rozwiązania Dekkera jest fakt, że testowanie i nadawanie wartości zmiennej stanowią dwie niezależne czynności. Gdy jeden proces testuje zmienną, to nie "wie" o tym, czy w tym samym czasie inny proces nie nadał tej zmiennej innej wartości. Chcąc wyeliminować tę przyczynę złożoności niektórzy autorzy postulują istnienie specjalnych instrukcji ułatwiających budowę aparatu synchronizacji procesów. I tak Lampson [11] zakłada istnienie rozkazu TSL n, którego treść jest następująca: jeśli (n) < 0 to (LR) + 2, jeśli (n) ≥ 0 to $-1 \longrightarrow n$ i (LR) + 1.

Część testowa każdego procesu, w której przeprowadza się badanie czy proces może wejść do swego rejonu krytycznego ma postać

```
TSL  LOCK          CRITICAL: -
SKO CRITICAL                 -
SKO * -2                     -
                   1 ——→ LOCK
```

Etykieta CRITICAL wskazuje początek ciągu rozkazów realizujących rejon krytyczny. Gdy zawartość pamięci LOCK jest ujemna proces "pętli się" wykonując rozkazy TSL LOCK i SKO* -2 do czasu, aż inny proces uczyni miejsce LOCK nieujemne, wówczas proces wejdzie do swego rejonu krytycznego ładując jednocześnie -1 do miejsca LOCK, blokując tym samym wejście do rejonu krytycznego innym procesom. Przed wyjściem z rejonu krytycznego

proces wykonuje operację 1⟶LOCK otwierając wejście do rejonu krytycznego.

Dijkstra [4] postuluje istnienie instrukcji swap (a, b), której wykonanie powoduje wymianę wartości zmiennych a i b. Sposób wykorzystania tego rozkazu ilustruje poniższy przykład zaczerpnięty z pracy [4] i tak zmodyfikowany, aby nie odbiegał w formie językowej od przykładów podanych w rozdziale 2.

Rozważmy n procesów komunikujących się ze sobą przez wspólną zmienną x. Z każdym procesem związana jest "prywatna" zmienna loc. W każdej chwili tylko jedna spośród n+1 zmiennych ma wartość zero; pozostałe zmienne mają wartość jeden. Proces jest w swoim rejonie krytycznym, jeśli jego zmienna loc ma wartość zero. Zakładając, że na początku żaden z procesów nie jest w rejonie krytycznym, czyli zmienna x ma wartość zero, struktura procesu ma postać

```
begin   integer loc; loc :=1;
        L: if loc ≠ 0 then begin swap (x, loc); goto L end
           critical section;
           swap (x, loc);
           remainder of cycle
end
```

Obydwie propozycje ułatwiają wprawdzie rozwiązanie problemu wykluczania, ale nie eliminują "aktywnego oczekiwania". Badanie czy można wejść do rejonu krytycznego angażuje czas procesora, który w tym czasie mógłby robić coś bardziej pożytecznego.

Aby uniknąć "aktywnego oczekiwania" Lampson podał następujące rozwiązanie.

Gdy proces stwierdzi, że nie może wejść do rejonu krytycznego umieszcza siebie na liście (wakeup list) procesów oczekujących na wejście do rejonu krytycznego następnie zawiesza swoje funkcjonowanie - zwalnia procesor. Proces opuszczający rejon krytyczny czyni to przez standardowy podprogram, który przegląda listę procesów oczekujących i "budzi" jeden z nich. Może się jednak zdarzyć, że w czasie gdy proces umieszcza się na liście

zostanie przerwany, co jest niedopuszczalne. Aby tego uniknąć podprogramy komunikujące się z listą powinny być nieprzerywalne. Lampson proponuje instrukcję PRO, której zadaniem jest zapewnienie, że w czasie krótkiego okresu czasu po wykonaniu PRO:

a) proces, który wykonał instrukcję PRO nie może utracić swego procesora lub być wywłaszczony[*] przez zegar

b) nie może być wykonana inna instrukcja PRO.

Jeśli inny procesor chce wykonać PRO musi odczekać aż zakończy się aktualne PRO.

Czas trwania instrukcji zależy od adresowania pośredniego, dlatego też czas trwania instrukcji PRO najlepiej mierzyć w liczbie sięgnięć do pamięci. Lampson twierdzi, że 15 sięgnięć do pamięci wystarczy.

Obie wyżej wspomniane próby rozwiązywania problemu synchronizacji są interesujące, niemniej jednak niemal powszechną aprobatę uzyskał pomysł Dijkstry [3] polegający na:

1. wprowadzeniu specjalnych zmiennych typu integer, zwanych semaforami

2. wprowadzeniu specjalnych operacji jednoargumentowych: P - operacji i V - operacji, których argumentami mogą być jedynie semafory.

Wartościami zmiennych typu semafor są liczby nieujemne. Gdy wartości semafora ograniczymy do zbioru $\{0, 1\}$, to semafor nazywamy binarnym. Semafory nie spełniające tego ograniczenia nazywamy ogólnymi.

V - operacja. Operacja V (S) zwiększa wartość semafora S o 1.

P - operacja. Operacja P(S) jest zdefiniowana następująco:
- jeśli $S \neq 0$, to P(S) powoduje zmniejszenie wartości S o 1
- jeśli $S = 0$, to operacja P(S) nie zmienia wartości S i nie jest zakończona do czasu, gdy inny proces nie wykona operacji V na tym samym semaforze.

---
*) W niniejszych materiałach słowo "wywłaszczenie" odpowiada angielskiemu terminowi preemption.

Istotę tych operacji stanowi fakt, że są one "niepodzielne" w następującym sensie: realizacja operacji V(S) wymaga 3 kroków

1. pobranie zmiennej z pamięci (I dostęp do pamięci)
2. zwiększenie zmiennej o jeden
3. umieszczenie zmiennej w pamięci (II dostęp do pamięci).

Otóż niepodzielność polega na tym, że po rozpoczęciu wykonywania operacji V(S) dostęp do zmiennej S powinien być zablokowany do chwili zakończenia operacji. Podobna uwaga odnosi się do operacji P. W danej chwili operację P może zainicjować więcej niż jeden proces, gdy semafor uzyska wartość 1 jedna z tych operacji zostanie zakończona.

Przy użyciu operacji V i P problem wykluczania staje się trywialny. Odpowiednik rozwiązania Dekkera wygląda następująco:

```
begin integer free; free :=1;
 parbegin
    process 1: begin L1: P (free); critical section 1; V (free)
                     remainder of cycle 1; goto L1
               end
    process 2: begin L2: P (free); critical section 2;V (free)
                     remainder of cycle 2; goto L2
               end
 parend
end
```

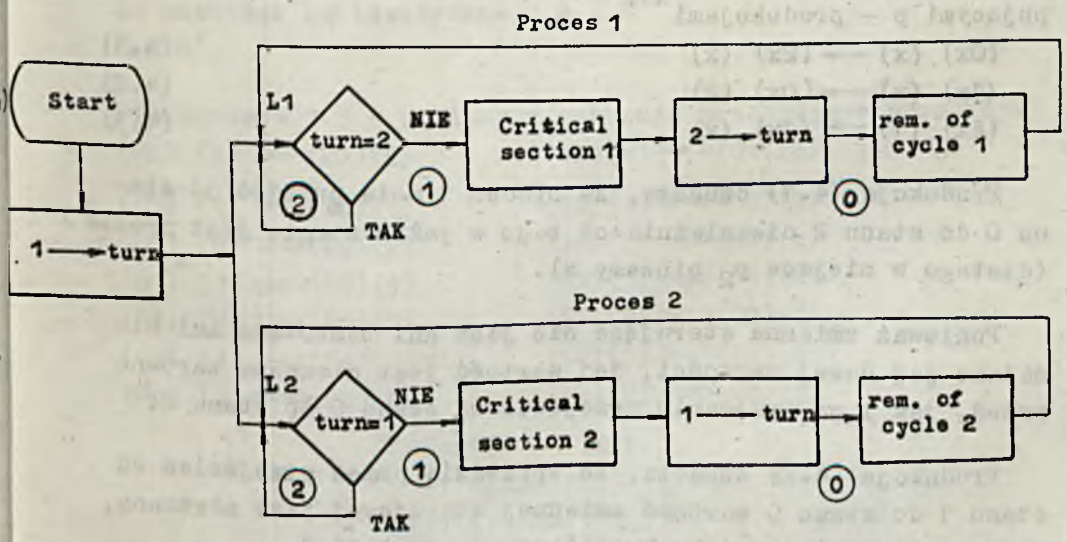
Prostotę rozwiązania uzyskano dzięki temu, że realizacja P i V wymaga rozwiązania problemu wykluczania na "niższym" poziomie, tj. gdy operujemy pojęciami "rozkaz maszyny", "dostęp do pamięci". Tak więc nastąpiło przesunięcie problemu wykluczania ze skali makro do skali mikro. Wydawać by się mogło, że istota problemu pozostaje bez zmian. Przy użyciu operacji P i V znacznie upraszcza się opis rozwiązania, a przez to problem synchronizacji sprowadza się do problemu realizacji tych dwóch operacji. Stanowi to jak gdyby "standaryzację" trudności występujących przy problemie synchronizacji.

Dijkstra [3] udowodnił, że każdy problem, który można rozwiązać stosując semafory ogólne, można również rozwiązać za pomocą semaforów binarnych.

Reasumując rozważania tego rozdziału można stwierdzić, że opierając się na elementarnych operacjach (TSL lub swap lub P i V) tworzy się aparat ułatwiający układanie programów synchronizujących. Pewne próby badania poprawności takich programów zostaną omówione w następnych rozdziałach.

4. FORMALIZM GILBERTA I CHANDLERA

Metoda ta zostanie przedstawiona na przykładzie analizy wersji 1 rozwiązania z rozdziału 2. Schemat czynnościowy wersji 1 wygląda następująco



Rys. 1

Każdy z procesów składa się z trzech części:

- część testowa, w której następuje badanie, czy proces może wejść do swego rejonu krytycznego
- rejon krytyczny
- pozostała część procesu

Proces może być w jednym z trzech stanów, stan 2 odpowiada części testowej, stan 1 - rejonowi krytycznemu, stan 0 - pozostałej części procesu.

Maszyna to trójka (P_1, P_2; v), gdzie P_i są procesami, a v jest zmienną sterującą.

Stan maszyny to układ (p_1 p_2) (d), gdzie p_1 jest stanem procesu 1, p_2 - stanem procesu 2, a d jest wartością zmiennej sterującej. Dla omawianego przykładu maszyna (P_1, P_2; turn) ma $3^2 \cdot 2 = 18$ stanów, ponieważ p_i może przyjmować wartości 0, 1, 2, a turn przyjmuje wartości 1, 2.

Poszczególne procesy opisuje się za pomocą tzw. p - produkcji ("partial rule"). Na przykład proces 1 można opisać następującymi p - produkcjami

$$(0x)\ (x) \longrightarrow (2x)\ (x) \tag{4.1}$$

$$(1x)\ (x) \longrightarrow (0x)\ (2) \tag{4.2}$$

$$(2x)\ (1) \longrightarrow (1x)\ (x) \tag{4.3}$$

Produkcja (4.1) oznacza, że proces 1 może przejść od stanu 0 do stanu 2 niezależnie od tego w jakim stanie jest proces 2 (dlatego w miejsce p_2 piszemy x).

Ponieważ zmienna sterująca nie jest ani testowana ani nie nadano jej nowej wartości, jej wartość jest nieznana zarówno przed, jak i po wykonaniu przejścia od stanu 0 do stanu 2.

Produkcja (4.2) oznacza, że wprawdzie przed przejściem od stanu 1 do stanu 0 wartość zmiennej sterującej jest nieznana, ale po wykonaniu tego kroku zmienna ma wartość 2.

Aby nie pisać p - produkcji nie wnoszących nic nowego wymaga się, aby p - produkcja zawierała albo zmianę stanu, albo nadanie wartości zmiennej. Dlatego p - produkcja

(2x) (2) ⟶ (2x) (x)

zośtała pominięta.

Analogicznie p - produkcje procesu 2 mają postać

(x0) (x) ⟶ (x2) (x) (4.4)

(x1) (x) ⟶ (x0) (1) (4.5)

(x2) (2) ⟶ (x1) (x) (4.6)

Zbiory p - produkcji generują produkcje opisujące działanie maszyny jako całości.

Sposób generowania produkcji zależy od tego, czy procesy działają jednocześnie czy niejednocześnie.

I. Działanie niejednoczesne

Niech R_1, R_2 będą p - produkcjami procesów P_1, P_2. Produkcja $S \Longrightarrow S'$ jest dopuszczalna, gdy

a) spełnia ograniczenie narzucone przez jedną z R_i

b) poza zmianami wartości wynikającymi z R_i wszystkie pozostałe wartości są identyczne w S i S'

c) $S \neq S'$

Dla procesu 1 p - produkcje generują następujące produkcje

(0x) (x) ⟶ (2x)(x) (1x)(x) ⟶ (0x)(2)

(00) (1) ⟹ (20)(1) (10)(1) ⟹ (00)(2)

(00) (2) ⟹ (20)(2) (10)(2) ⟹ (00)(2)

(01) (1) ⟹ (21)(1) (11)(1) ⟹ (01)(2)

(01) (2) ⟹ (21)(2) (11)(2) ⟹ (01)(2)

(02) (1) ⟹ (22)(1) (12)(1) ⟹ (02)(2)

(02) (2) ⟹ (22)(2) (12)(2) ⟹ (02)(2)

(2x)(1) ⟶ (1x)(x)

(20)(1) ⟹ (10)(1)

(21)(1) ⟹ (11)(1)

(22)(1) ⟹ (12)(1)

Analogicznie można wypisać produkcje dla procesu 2.

II. Działanie jednoczesne

Niech będą dane p - produkcje

$(p_1\ x)(a) \longrightarrow (p_1'\ x)(c)$ dla procesu 1

$(x\ p_2)(b) \longrightarrow (x\ p_2')(d)$ dla procesu 2

Taka para nie generuje produkcji jeśli jest spełniony warunek

$W1: a \neq b\ \ (a \neq x\ \text{i}\ b \neq x)\ \text{lub}\ c \neq d\ \ (c \neq x\ \text{i}\ d \neq x)$

Gdy warunek W1 nie jest spełniony, to otrzymujemy produkcję

$(p_1\ p_2)(w) \Longrightarrow (p_1',\ p_2')\ (z)$

gdzie

$$w = \begin{cases} a & \text{jeśli } a \neq x \\ b & \text{jeśli } b \neq x \\ 1,\ 2 & \text{jeśli } a = b = x \end{cases}$$

(tworzy się dwie produkcje dla zmiennej = 1 i zmiennej = 2)

$$z = \begin{cases} c & \text{jeśli } c \neq x \\ d & \text{jeśli } d \neq x \\ w & \text{jeśli } c = d = x \end{cases}$$

Stosując tę regułę otrzymujemy dalsze produkcje

R_1 \ R_2	(x0) (x)→(x2)(x)	(x1)(x)→(x0) (1)	(x2) (2)→(x1) (x)
(0x) (x)→(2x)(x)	(00)(1)⇒(22)(1) (00)(2)⇒(22)(2)	(01)(1)⇒(20) (1) (01)(2)⇒(20) (1)	(02) (2)⇒(21) (2)
(1x) (x)→(0x)(2)	(10)(1)⇒(02)(2) (10)(2)⇒(02)(2)		(12) (2)⇒(01) (2)
(2x) (1)→(1x)(x)	(20)(1)⇒(12)(1)	(21)(1)⇒(10) (1)	

Łącząc uzyskane produkcje otrzymujemy następujący opis działania maszyny.

Tabela 1

R_1 \ R_2	(x0)(x)→(x2)(x)	(x1)(x)→(x0)(1)	(x2)(2)→(x1)(x)	NIE DZIAŁA
(0x)(x)→(2x)(x)	(00)(1)⇒(22)(1) (00)(2)⇒(22)(2)	(01)(1)⇒(20)(1) (01)(2)⇒(20)(1)	(02)(2)⇒(21)(2)	(00)(1)⇒(20)(1) (00)(2)⇒(20)(2) (01)(1)⇒(21)(1) (01)(2)⇒(21)(2) (02)(1)⇒(22)(1) (02)(2)⇒(22)(2)
(1x)(x)→(0x)(2)	(10)(1)⇒(02)(2) (10)(2)⇒(02)(2)		(12)(2)⇒(01)(2)	(10)(1)⇒(00)(2) (10)(2)⇒(00)(2) (11)(1)⇒(01)(2) (11)(2)⇒(01)(2) (12)(1)⇒(02)(2) (12)(2)⇒(02)(2)
(2x)(1)→(1x)(x)	(20)(1)⇒(12)(1)	(21)(1)⇒(10)(1)		(20)(1)⇒(10)(1) (21)(1)⇒(11)(1) (22)(1)⇒(12)(1)
NIE DZIAŁA	(00)(1)⇒(02)(1) (00)(2)⇒(02)(2) (10)(1)⇒(12)(1) (10)(2)⇒(12)(2) (20)(1)⇒(22)(1) (20)(2)⇒(22)(2)	(01)(1)⇒(00)(1) (01)(2)⇒(00)(1) (11)(1)⇒(10)(1) (11)(2)⇒(10)(1) (21)(1)⇒(20)(1) (21)(2)⇒(20)(1)	(02)(2)⇒(01)(2) (12)(2)⇒(11)(2) (22)(2)⇒(12)(2)	

Na podstawie tej tabeli można narysować graf przejść uwzględniając fakt, że stanem początkowym jest stan (22)(1).

Stanem zabronionym nazywamy stan, w którym obydwa procesy są w swych rejonach krytycznych. A zatem zbiór stanów zabronionych $E = \{(11)(1), (11)(2)\}$.

Niech S_0 będzie stanem początkowym maszyny. Jeśli istnieje ciąg produkcji $S_0 \Longrightarrow S_1$, $S_1 \Longrightarrow S_2, \ldots, S_{k-1} \Longrightarrow S_k$, to piszemy $S_0 \overset{+}{\Longrightarrow} S_k$ i mówimy, że stan S_k jest osiągalny.

Niech Q będzie zbiorem wszystkich możliwych stanów maszyny, wówczas zbiór produkcji T dzieli zbiór Q na dwa zbiory rozłączne

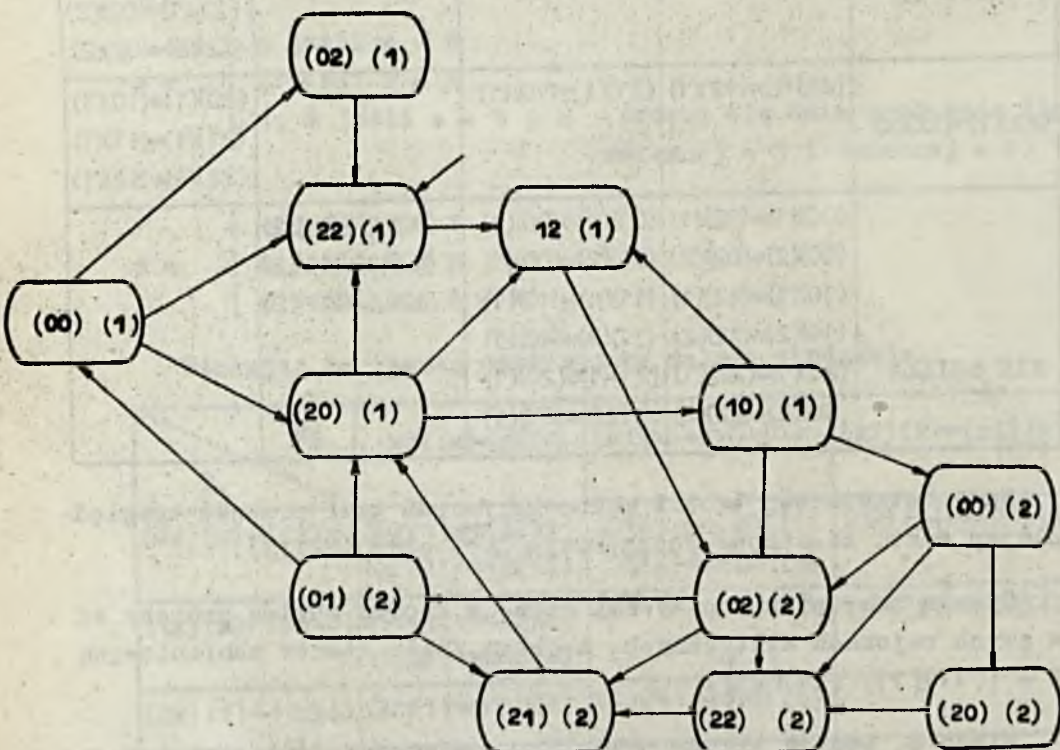
$$Q = A\ (T) \cup \quad A_1\ (T)$$

gdzie

$$A\ (T) = \{S \in Q : S_0 \overset{+}{\Longrightarrow} S\} \cup \{S_0\}$$
$$A_1(T) = \{S \in Q : S \notin A\ (T)\}$$

Maszyna jest poprawna gdy $E \cap A\ (T) = 0$.

Sprawdźmy czy rozważana maszyna jest poprawna. W tym celu należy sprawdzić, czy zachodzi relacja $S_0 \overset{+}{\Longrightarrow} (11)(1)$ lub $S_0 \overset{+}{\Longrightarrow} (11)(2)$. Na podstawie tabeli 1 stwierdzamy, że $(21)(1) \Longrightarrow (11)(1)$ i $(12)(2) \Longrightarrow (11)(2)$.



Rys. 2

"Cofając" się w ten sposób otrzymujemy

$$(01)\ (1) \overset{+}{\Longrightarrow} (11)(1) \quad \text{i} \quad (10)(2) \overset{+}{\Longrightarrow} (11)(2)$$

Łatwo sprawdzić, że żaden ze stanów (01)(1), (10)(2) nie jest osiągalny, a więc nasza maszyna jest poprawna.

Przedstawiona metoda umożliwia przeprowadzenie analizy na maszynie cyfrowej. I tak autorzy pracy [5] podają, że przy założeniu niejednoczesności przeanalizowano maszynę składającą się z 3 procesów i mającą 1900 osiągalnych stanów. Analiza trwała 2,5 minuty na maszynie IBM 360/65.

5. FORMALIZM HABERMANNA

Interesującą próbę stwierdzenia poprawności synchronizacji procesów podał Habermann [7]. Synchronizację opisuje on za pomocą operacji wait (s) i signal (s), których znaczenie jest takie samo jak operacji P (S) i V(S).

Zatem mamy dwie operacje wait (s) i signal (s), rozważamy poza tym stan synchronizacji opisany stałą c [s] oraz zmiennymi nw(s), ns(s), np(s) z początkowymi wartościami zerowymi, które oznaczają:

- nw(s) - liczba wykonań operacji wait (s)
- ns(s) - liczba wykonań operacji signal (s)
- np(s) - liczba wykonań instrukcji znajdującej się za instrukcją wait (s).

Efektem wykonania wait (s) jest:

nw(s) := nw(s) + 1; <u>if</u> nw(s) ≤ c [s]+ ns(s)
<u>then</u> np(s) := np(s) + 1

Wykonanie wait (s) nie powoduje opóźnienia, gdy jest spełniony warunek nw (s) ≤ c[s] + ns (s)

Efektem wykonania signal (s) jest:

<u>if</u> nw (s) > c[s]+ ns(s) <u>then</u> np (s) := np(s) + 1;
ns(s) := ns (s) + 1

Habermann udowodnił następujące twierdzenie:

Twierdzenie.

Relacja

$$np(s) = \min(nw(s),\ c[s] + ns(s)) \qquad (H)$$

jest nieznienniocza zarówno względem operacji wait (s) , jak i operacji signal (s) .

Opierając się na tym twierdzeniu można dowodzić, czy dana synchronizacja jest poprawna.

Przykład 1. Rozważmy synchronizację procesów w wypadku problemu wykluczania.

Rejony krytyczne procesów komunikujące się ze wspólnym zbiorem danych s można zaprogramować w postaci

wait (s); Z1;..., Zn; signal (s)

gdzie Z1, Z2,..., Zn są operacjami na zbiorze s.

Zakłada się, że $c[s] = 1$.

Przy dowodzeniu poprawności funkcjonowania rejonu krytycznego zakłada się, że niemożliwe jest wejście lub opuszczenie rejonu krytycznego pomijając operacje wait i signal.

Rejon krytyczny funkcjonuje poprawnie, gdy

a) tylko jeden rejon krytyczny może być wykonywany w danej chwili

b) żaden proces nie będzie opóźniony, jeśli żaden z nich nie znajduje się w rejonie krytycznym

Dowód a. Z relacji (H) wynika

$$np(s) \leqslant 1 + ns(s)$$

tzn. liczba wejść do rejonu krytycznego nie przekracza liczby wyjść z rejonu krytycznego zwiększonej o jeden

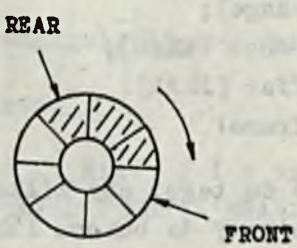
$$np(s) - ns(s) \leqslant 1$$

Dowód b. W wypadku istnienia procesów opóźnianych zachodzi $nw(s) > 1 + ns(s)$ czyli z relacji (H) wynika, że

np (s) = 1 + ns (s)

Z drugiej strony łatwo zauważyć, że gdy żaden proces nie jest w rejonie krytycznym, to np (s) = ns (s).

Przykład 2. Rozważmy dwa procesy asynchroniczne, nadajnik S i odbiornik R komunikujące się przez bufor pierścieniowy (rys. 3).



Rys. 3

Z buforem związane są dwie zmienne FRONT i REAR. Zmienna FRONT wskazuje pierwsze wolne miejsce, natomiast zmienna REAR - pierwsze zajęte miejsce. Stan początkowy buforu jest taki, że succ (REAR) = FRONT, gdzie succ jest funkcją następnika. Wielkość buforu określa zmienna "bufsize".

Po przygotowaniu przez S komunikatu d zostaje on umieszczony w buforze

```
deposit: buffer [FRONT] := d;
         FRONT:= succ (FRONT)
```

Przed przetworzeniem przez R komunikat r zostaje pobrany z buforu

```
accept: REAR := succ (REAR)
        r := buffer [REAR]
```

Synchronizację ze względu na przepełnienie buforu przy operacji deposit realizujemy wprowadzając zmienną "frame", przy czym c [frame] = bufsize.

Synchronizację ze względu na pusty bufor przy operacji accept realizujemy wprowadzając zmienną "message", przy czym c [message] = 0

```
deposit: wait (frame);
         buffer [FRONT] := d;
         FRONT := succ (FRONT);
         signal (message)
accept:  wait (message);
         REAR := succ (REAR);
         r := buffer [REAR];
         signal (frame)
```

Nie można dopuścić do tego, aby kilka nadajników umieszczało jednocześnie komunikaty do buforu lub też kilka odbiorników pobierało komunikaty. Dlatego operacje deposit i accept trzeba zaprogramować jako rejony krytyczne. Uczynimy to za pomocą zmiennych "input" i "output", gdzie c [input] = c [output] = 1.

Ostateczna wersja procedur jest następująca:

```
procedure deposit (d);
begin
   wait (input);
      wait (frame);
      buffer [FRONT] := d;
      FRONT := succ (FRONT);
      signal (message);
   signal (input)
end

procedure accept (r);
begin
   wait (output);
      wait (message);
      REAR := succ (REAR);
      r := buffer [REAR];
      signal (frame);
   signal (output)
end
```

Posługując się relacją (H) można udowodnić pewne własności tak rozwiązanej synchronizacji

1. Przy jednoczesnym umieszczaniu i pobieraniu komunikatu z buforu zmienne FRONT i REAR nie wskazują tego samego miejsca.

2. Nie powstanie sytuacja kiedy:
   - liczba umieszczeń przekracza liczbę pobrań zwiększoną o bufsize
   - liczba pobrań przekracza liczbę umieszczeń

3. Nie powstanie blokada.

Udowodnimy własność 1. Niech f i u oznaczają liczby wykonań operacji succ na zmiennych FRONT i REAR.

Gdy komunikat jest umieszczany, tzn. gdy wykonywana jest operacja buffer [FRONT] := d
wówczas

$$f = ns\ (message) \tag{5.1}$$

W tym czasie zachodzi

$$ns\ (message) = np\ (frame) - 1 \leq bufsize + ns\ (frame) - 1 \tag{5.2}$$

na mocy treści procedury deposit i relacji (H).

Gdy komunikat jest pobierany, tzn. gdy r := buffer [REAR] wówczas

$$u = np\ (message) \tag{5.3}$$

oraz

$$np\ (message) = ns\ (frame) + 1 \leq ns\ (message) \tag{5.4}$$

na mocy treści procedury accept i relacji (H).

Uwzględniając (5.1) i (5.3) wzory (5.2) i (5.4) można napisać w postaci

$$f = np\ (frame) - 1 \leq bufsize + ns\ (frame) - 1 \tag{5.5}$$

$$u = ns\ (frame) + 1 \leq f \tag{5.6}$$

Gdy umieszczanie i pobieranie odbywa się jednocześnie, wówczas pisząc wzór (5.5) w postaci

$$f \leqslant bufsize + (ns\,(frame) + 1) - 2 \tag{5.7}$$

i uwzględniająo w nim wzór (5.6) otrzymujemy

$$f \leqslant bufsize + u - 2 \tag{5.8}$$

uwzględniająo w tym wzorze wzór (5.6) mamy

$$u \leqslant f \leqslant bufsize + u - 2$$

ozyli

$$0 \leqslant f - u \leqslant bufsize - 2 \tag{5.9}$$

Zmienne FRONT i REAR wskazują ten sam obszar buforu, gdy

$$u = f + 1\ (mod\ bufsize) \tag{5.10}$$

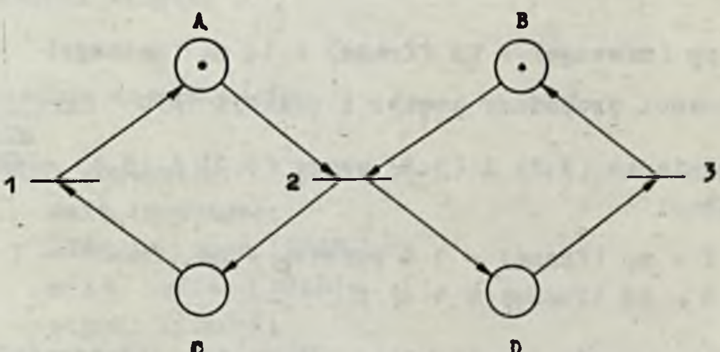
Ponieważ wzory (5.9) i (5.10) są sprzeczne, FRONT i REAR nie mogą wskazywać tego samego obszaru.

6. SIECI PETRI

Opis systemu liczącego w formaliźmie sieci Petri umożliwia dynamiczną analizę zachowania się systemu.

Sieć Petri jest grafem zorientowanym mającym dwa typy wierzchołków: wierzchołki zwane miejscami i wierzchołki zwane przejściami.

Przykład 1. Rozważmy sieć przedstawioną na rys. 4.

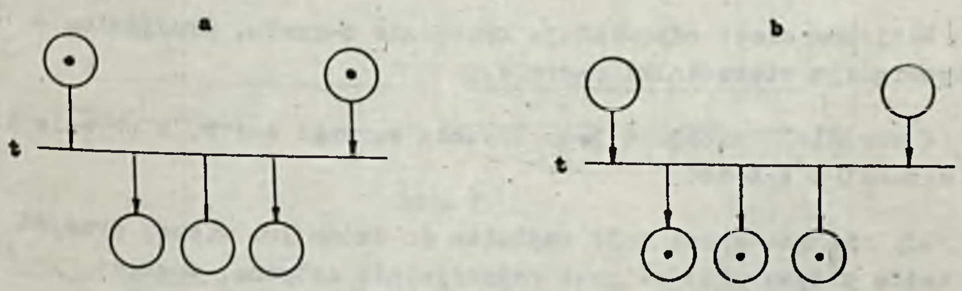


Rys. 4

A, B, C, D - miejsca
1, 2, 3 - przejścia
Miejsca A i B są wejściami przejścia 2
Miejsca C i D są wyjściami przejścia 2.

Znakowaniem sieci nazywamy funkcję
$f : P \longrightarrow N$, gdzie P jest zbiorem miejsc
a $N = \{0, 1, 2, ...\}$
W przykładzie 1 $f(A) = 1$, $f(C) = 0$
Przejście t jest potencjalnie aktywne, jeśli dla każdego wejścia p przejścia t zachodzi $f(p) > 0$.

Na przejściu potencjalnie aktywnym można wykonać operację strzelania. Efekt operacji strzelania ilustruje rys. 5.



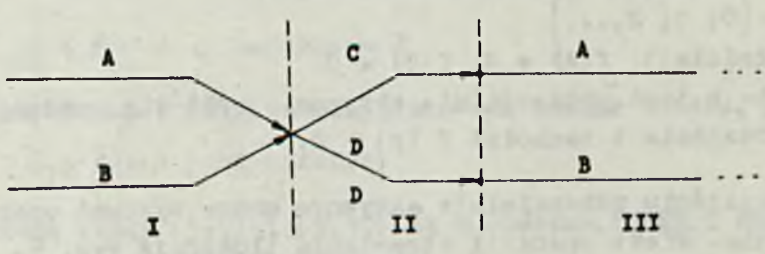
Rys. 5
a/ stan sieci przed wykonaniem operacji strzelania,
b stan sieci po wykonaniu operacji strzelania

Przy opisywaniu systemu za pomocą sieci miejscu odpowiada warunek. Gdy miejsce zawiera kropkę oznacza to, że warunek jest spełniony.

Stan systemu jest określony przez sieć wraz ze znakowaniem, co oznacza, że zachodzą jednocześnie wszystkie warunki odpowiadające miejscom oznakowanym. Wykonanie operacji strzelania powoduje przejście do nowego znakowania, które określa nowy stan systemu. Przy czym zakłada się, że operacja strzelania

nie wymaga czasu, przejście od jednego stanu do drugiego jest natychmiastowe. Czas jest związany z zachodzeniem danego warunku (czas pobytu kropki w danym miejscu).

Historię symulacji sieci opisuje się za pomocą o-grafów. O-graf sieci z przykładu 1 przedstawia rys. 6.



Rys. 6.

Miejscom sieci odpowiadają krawędzie o-grafu, przejściom - odpowiadają wierzchołki o-grafu.

W okresie I zachodzą jednocześnie warunki A i B, w okresie II - warunki C i D itd.

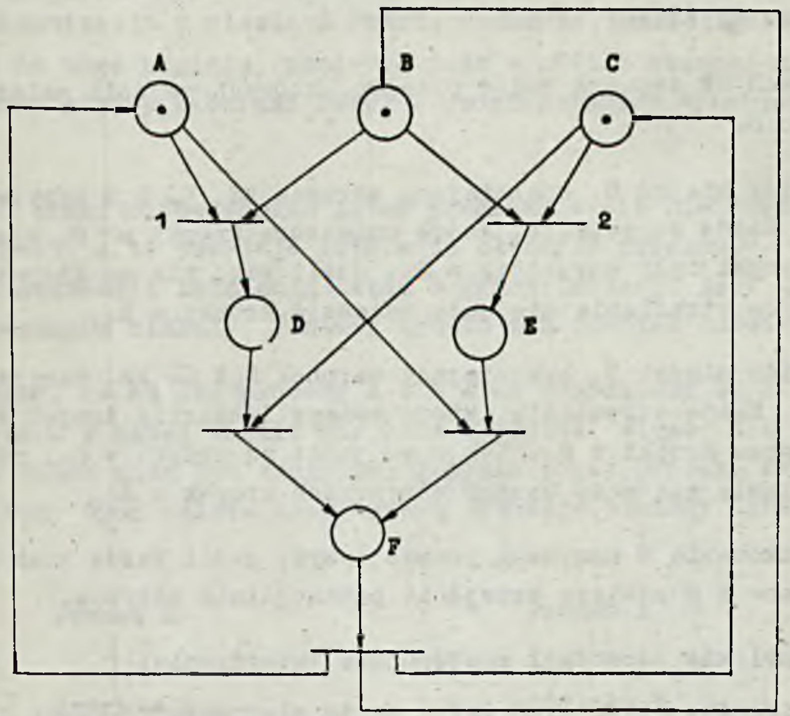
Gdy miejsce sieci jest wejściem do dwóch lub więcej przejść i każde z tych przejść jest potencjalnie aktywne, wówczas przejścia te nazywamy konfliktowymi.

Przykład 2. Rozważmy sieć przedstawioną na rys. 7.

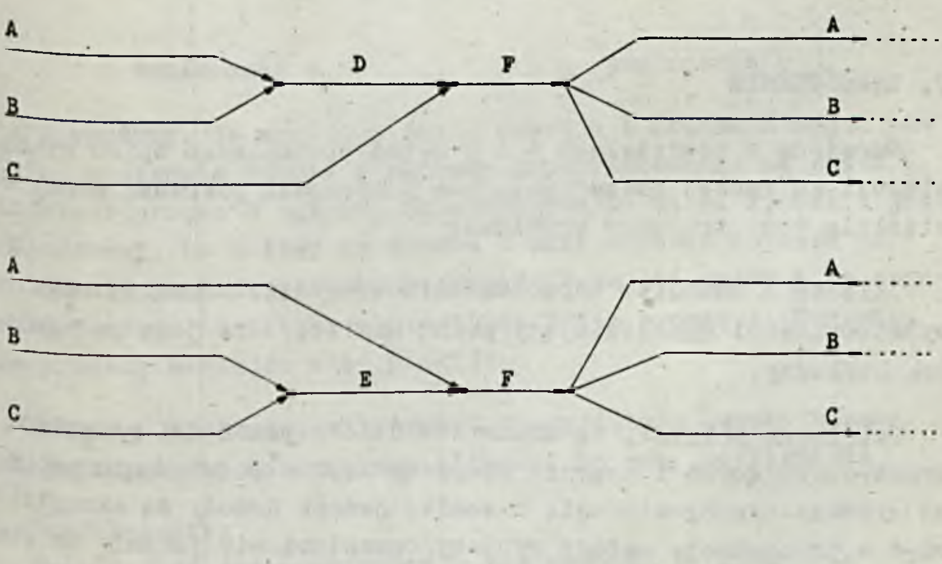
Przejścia 1, 2 są konfliktowe. W takiej sytuacji tylko na jednym przejściu można wykonać operację strzelania. O-graf sieci z rys. 7 przedstawiono na rys. 8.

Interesująca byłaby możliwość badania dynamicznych własności systemu opisanego za pomocą sieci Petri na podstawie samej struktury sieci (z pominięciem symulacji). Badania w tym kierunku przeprowadził Tourlakis [13].

Niech S będzie podzbiorem miejsc sieci Petri.



Rys. 7



Rys. 8

Niech RS oznacza zbiór przejść, dla których elementy zbioru S są wyjściami.

Niech SR oznacza zbiór przejść, których wejścia należą do zbioru S.

Zbiór miejsc S_1 spełniający warunek $RS_1 \subseteq S_1R$ nazywamy blokadą. Każde strzelanie, które umieszcza kropki w S_1 wymaga, aby kropki były uprzednio w S_1. Jeśli więc nie ma kropek w S_1, to żadne strzelanie nie może umieścić kropki w S_1.

Zbiór miejsc S_2 spełniający warunek $S_2R \subseteq RS_2$ nazywamy pułapką. Każde strzelanie, które wymaga istnienia kropek w S_2 umieszcza kropki w S_2. Tak więc jeśli są kropki w S_2, to żadne strzelanie nie może usunąć wszystkich kropek z S_2.

Znakowanie M nazywamy pseudo żywym, jeśli każde znakowanie uzyskane z M zawiera przejście potencjalnie aktywne.

Tourlakis udowodnił następujące twierdzenie:

Twierdzenie. Niech sieć Petri ma tę własność, że każda blokada zawiera pułapkę. Wówczas każde znakowanie, które umieszcza co najmniej jedną kropkę w każdej pułapce jest pseudo żywe.

7. ZAKOŃCZENIE

Omówione w rozdziałach 4 i 5 metody formalnego opisu synchronizacji są raczej niezadowalające i stanowią pierwsze próby rozwiązania tego trudnego problemu.

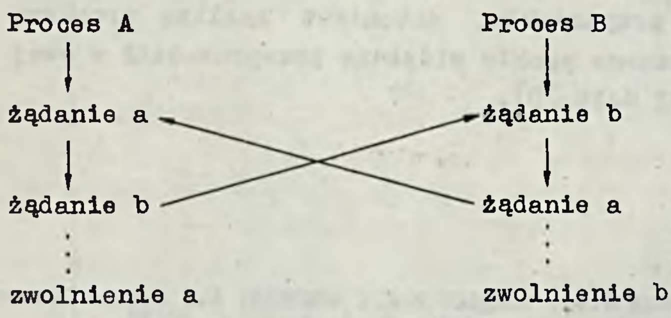
Gilbert i Chandler zaproponowali wprawdzie formalny opis synchronizacji umożliwiający pełną analizę, ale jest on bardzo pracochłonny.

Habermann pokazał, że można dowodzić poprawności programów synchronizujących i uczynił pierwszy krok w rozwiązaniu problemu syntezy synchronizacji; niemniej jednak dowody są skomplikowane a prezentacja metody syntezy ogranicza się jedynie do rozwiązania kilku przykładów.

Niektórzy autorzy wiążą duże nadzieje na rozwiązanie problemu synchronizacji z sieciami Petri. Jednakże trudno ustosunkować się do tego poglądu, ponieważ jest w chwili obecnej niejasny związek między sieciami Petri a programowaniem synchronizacji.

Wobec braku odpowiednich metod programuje się nie dowodząc poprawności, a to powoduje istnienie błędnych programów. Jedną z konsekwencji istnienia błędu w synchronizacji jest możliwość powstania blokady. Poniżej krótko ten problem omówimy.

Załóżmy, że są dwa procesy A i B oraz dwa zasoby a, b. Z danego zasobu w danej chwili nie może korzystać więcej niż jeden proces. Zasób może być zwolniony jedynie przez proces, który go zajął. Przy tych założeniach typową sytuację blokady ilustruje rysunek



Przypuśćmy, że proces A zajął zasób a i proces B zajął zasób b, następnie proces A zgłasza zapotrzebowanie na zasób b, natomiast proces B zgłasza zapotrzebowanie na a. Proces A jest zablokowany, bo dostęp do zasobu b może uzyskać dopiero po zwolnieniu go przez proces B. Proces B zwolni zasób b po uzyskaniu dostępu do zasobu a, zajętego przez proces A. Tak więc oba procesy nawzajem się blokują.

W każdym systemie, w którym procesy dzielą zasoby trzeba ustosunkować się do problemu blokady. Są trzy możliwości:

1. Zapobieganie.
System jest zaprojektowany w ten sposób, że powstanie blokady jest niemożliwe.

2. Wykrywanie.
Stan blokady może powstać, ale jest automatycznie wykrywany. Wyjście z tego stanu polega na wymuszeniu zakończenia zablokowanych procesów lub też uwolnienia zasobów przypisanych zablokowanym procesom.
3. Katastrofa.
Blokada może powstać i nie jest automatycznie wykrywana. Ocenę czy powstał stan blokady i ewentualne środki zaradcze pozostawiono w gestii operatora.

W [1] Coffman, Elphick, Shoshani sformułowali warunki konieczne do tego, aby w systemie mogła powstać blokada. Jedną z metod zabezpieczenia się przed powstaniem blokady w systemie jest zaprojektowanie systemu w ten sposób, aby nie był spełniony jeden z wyżej wspomnianych warunków. Taką metodę zastosowali Murphy [12], Mavender [8] i Habermann [6]. Prace tych autorów mają charakter przyczynków, natomiast analizę problemu blokady z teoretycznego punktu widzenia przeprowadził w swej rozprawie doktorskiej Holt [9].

Literatura

[1] COFFMAN E.G., ELPHICK M.J.. SHOSHANI A.: System Deadlocks, Computing Surveys 1971, t. 3, nr 2, s. 67-78.

[2] DIJKSTRA E.W.: Solution of a Problem in Concurrent Programming Control, Comm. ACM 1965, t. 8, nr 9, s. 569.

[3] DIJKSTRA E.W.: Co-operating Sequential Processes, Programming Languages. F. Genuys (Ed.), Academic Press, New York 1968, s. 43-112.

[4] DIJKSTRA E.W.: Hierarchical Ordering of Sequential Processes, Acta Informatica, 1971, nr 1, s. 115-38.

[5] GILBERT P., CHANDLER W.J.: Interference Between Communicating Parallel Processes, Comm. ACM 1972, t. 15, nr 6, s. 427-37.

[6] HABERMANN N.A.: Prevention of System Deadlocks, Comm. ACM 1969, t. 12, nr 7, s. 373-7, 385.

[7] HABERMANN N.A.: Synchronization of Communicating Processes, Comm. ACM 1973, t. 15, nr 3, s. 171-6.

[8] HAVENDER J.W.: Avoiding Deadlock in Multitasking Systems, IBM Systems J. 1968, t. 7, nr 2, s. 74-84.

[9] HOLT R.C.: Some Deadlock Properties of Computer Systems, Computing Surveys 1972, t. 4, nr 3, s. 179-96.

[10] KNUTH D.: Additional Comments on a Problem in Concurrent Programming Control, Comm. ACM 1966, t. 9, nr 5, s. 321-2.

[11] LAMPSON B.W.: A Scheduling Philosophy for Multiprocessing Systems, Comm. ACM 1968, t. 11, nr 5, s. 347-60.

[12] MURPHY J.E.: Resource Allocation with Interlock Detection in a Multitask System, Proc. AFIPS 1968, t. 33, nr 2, s. 1169-76.

[13] TOURLAKIS G.: Petri Nets. Topics in Operating Systems Revisited, Ed. D. Tsichritzis, 1-24, University of Toronto. Dep. of Computer Science. Techn. Rep. 1971, nr 38.

МЕТОДЫ ПРЕДСТАВЛЕНИЯ СИНХРОНИЗАЦИИ ПРОЦЕССОВ

Резюме

Главное затруднение в понимании функционирования сложной вычислительной системы заключается в проблеме синхронизации процессов.

В настоящем труде дается обзор методов решения проблемы синхронизации:

1. Програмное решение Дайкотры, в котором использованы лишь логические и целые переменные, а также не предлагаются никакие специальные инструкции языка высокого уровня.
2. Формализм Джильберта и Чандлера, позволяющий автоматизировать проверку решения.
3. Предложение Лампсона, определяющее специальные инструкции TSL и PRO, облегчающие синхронизацию параллельных процессов.
4. Решение проблемы синхронизации при использовании семафоров и операций P и V Дайкстры.
5. Формализм Габерманна, в котором можно доказывать свойства решения данной синхронизации.

DESCRIPTION METHODS OF PROCESS SYNCHRONIZATION

Summary

Difficulties in gaining an understanding of the behaviour of a complex computer system come from the problem of process synchronization.

This paper presents a survey of different approaches to the synchronization problem.

1. The programmed solution of Dijkstra in which only logical and integer variables are used; there is no extension to the usual statements of high-level languages.
2. The analysis formalism of Gilbert and Chandler allowing a mechanical proof procedure which will either verify or discredit any solution.
3. The proposal of Lampson who defines the special machine instructions TSL and PRO to facilitate synchronization between concurrent processes.
4. The solution to the process synchronization by means of semaphores and P and V operations introduced by Dijkstra.
5. The formalism of Habermann in which we can precisely prove properties of a given synchronization.

[illegible]

ALGORYTMY, Zeszyt Specjalny, 1974, 53-67

681.322.06:
061.3/438/"1973"

SYNCHRONIZACJA PROCESÓW W MASZYNIE CYFROWEJ

Stanisław CHROBOT
Wojskowa Akademia Techniczna

Pracę złożono 10.I.1974

Na podstawie ogólnej definicji procesu wprowadzono pojęcia procesu programowanego i monitora. Proces programowany jest szeregową kombinacją procesów podstawowych odpowiadających instrukcjom programu. Monitor jest procesem synchronizującym dla procesów programowanych i zewnętrznych. Przedstawiono dwa algorytmy cyklów rozkazowych procesora. Pierwszy opisuje sprzętowy monitor o minimalnym i kompletnym zbiorze funkcji synchronizujących. Drugi jest algorytmem cyklu rozkazowego procesora wirtualnego jako własnego procesora każdego procesu programowanego.

Spis treści

1. MONITOR I PROCES PROGRAMOWANY

Aby ogarnąć złożoność problemów przy zachowaniu się procesora wyposażonego w układ przerwań definiuje się pojęcie procesu. Jako intuicyjną podstawę podziału czynności na procesy

przyjmuje się zasadę "ciągłości" licznika rozkazów. Zgodnie z tą zasadą jako następną operację procesu przyjmuje się wykonanie rozkazu o adresie wskazanym przez licznik rozkazów wyznaczony w trakcie wykonywania poprzedniego rozkazu tego procesu. Eliminuje się więc jako obce dla procesu te ciągi rozkazów, które zostały wywołane przez niezwiązane z tym procesem przerwania.

Przy rozpatrywaniu własności procesów trudno uzyskać właściwą ogólność opisu działania samego procesora, gdyż efekt działania procesu zależy od wymienialnego programu. Z drugiej strony trudno rozdzielić operacje wchodzące w skład procesów od operacji związanych z przełączaniem procesora między procesy.

Rozwiązaniem pozwalającym wyeliminować te trudności może być podejście Horninga i Randella przedstawione w [1].

Pozwala ono z jednej strony dekomponować procesy, z drugiej - pozwala traktować jako proces kombinację procesów składowych. Jedną z możliwych form kombinacji jest k o m b i n a c j a s z e r e g o w a. Tworzy ją zbiór procesów z tą własnością, że w każdym stanie najwyżej jeden proces jest aktywny.

Dalej Horning i Randell sugerują, aby wykonywanie konwencjonalnego programu sekwencyjnego potraktować jako kombinację szeregową podstawowych procesów, z których każdy jest zdefiniowany przez instrukcje programu. Ta kombinacja współdziała z procesem sterującym, który określa instrukcję definiującą kolejny proces podstawowy. Proces sterujący i kombinacja procesów podstawowych komunikuje się ze sobą przez zestaw wspólnych zmiennych, zwanych s t a t u s e m p r o g r a m u, w którym centralną rolę gra licznik rozkazów. Ogólnie proces sterujący może współdziałać z kilkoma kombinacjami procesów podstawowych (z których każda posiada własny status), używającymi tego samego procesora w różnym czasie i synchronizującymi się wzajemnie

Proces sterujący nazywać będziemy dalej m o n i t o r e m. Kombinację procesów podstawowych komunikującą się z monitorem za pomocą ustalonego zbioru zmiennych nazwiemy p r o c e s e m p r o g r a m o w a n y m, zaś ten zbiór zmiennych - s t a t u s e m p r o c e s u.

Istotą tego podejścia jest, zdaniem autora, potraktowanie monitora nawet w przypadku, gdy jest on realizowany metodą układowo-programową, nie na równi z procesem programowym, ale z procesem podstawowym. W stwierdzeniu tym zawarty jest więc postulat, aby działanie monitora traktować jako składnik cyklu rozkazowego procesora, a co za tym idzie, aby jego realizacja była całkowicie układowa.

2. MONITOR MINIMALNY

Aby powyższy postulat uczynić realnym, należy określić minimalny, ale kompletny zbiór funkcji monitora. Pomijając funkcje sterowania sekwencyjnego, którymi się tu zajmować nie będziemy, przyjmiemy jako kompletny zbiór funkcji monitora zestaw operacji monitora bazowego opisany przez Hansena w [2].

W skład monitora bazowego wchodzą operacje:

- inicjowania i kończenia procesów;
- wait i signal na semaforach;
- await i cause na zdarzeniach (event variable).

Hansen pokazuje jak na ich bazie zrealizować konstrukcje języka programowania wysokiego poziomu takie jak proste i warunkowe regiony krytyczne oraz zdania współbieżne, które w sposób wygodny i bezpieczny pozwalają pisać programy dla współistniejących procesów programowanych [2], [3].

Okazuje się, że konstrukcje te można zrealizować na bazie dużo skromniejszego monitora wykonującego jedynie operacje wait i signal.

Poniżej przedstawimy sposób realizacji regionów krytycznych, w których stosowane są operacje await i cause.

Posłużymy się w tym celu językiem PASCAL [4] rozszerzonym o notację dla prostych regionów krytycznych [2] w formie: region v do S.

Zdefiniujemy najpierw typ zmiennej synchronizującej zwany bramką równoległą i trzy procedury, które jako jedyne mogą działać na zmiennej tego typu (p. Algorytm 1). Składnik "w" zmiennej B będącej parametrem tych procedur gra więc rolę "zamka", od którego stanu zależy, czy proces przez nią "przechodzący" będzie "przepuszczony" czy "zatrzymany", zaś składnik "l" - rolę licznika "zatrzymanych", gdy bramka jest zamknięta. Po otwarciu "zamka" zwalniane są równolegle wszystkie procesy zatrzymane, co tłumaczy nazwę bramki. W tym kontekście semafor można by nazwać bramką szeregową.

ALGORYTM 1
"OPERACJE SYNCHRONIZACJI RÓWNOLEGŁEJ"

```
type  bramka równoległa =
      record
         b:shared record w: boolean; l: integer end;
         s: semafor
      end
      {początkowo l=s=0}
procedure przejdź (var B: bramka równoległa);
   begin with B do
      begin region b do
         with b do if w then signal (s) else l:=l+1;
         wait (s)
      end;
   end
procedure zamknij (var B: bramka równoległa);
   begin with B do
      region b do with b do w:=false
   end
```

```
procedure otwórz (var B: bramka równoległa);
  begin with B do
    region b do with b do
    begin
      w:= true;
      while l > 0 do begin l:= l- 1; signal (s) end;
    end;
  end
```

Wówczas użycie procedur await i cause wewnątrz regionów krytycznych można zastąpić następująco:

```
region v do
begin
  while not W do
  await (e);
  S1;
end
```

```
with v do
begin
  wait (mutex);
  while not W do
  begin
    zamknij (B);
    signal (mutex);
    przejdź (B);
    wait (mutex)
  end;
  S1; signal (mutex)
end
```

```
region v do
begin
  S2;
  cause (e)
end
```

```
with v do
begin
  wait (mutex);
  S2;
  otwórz (B);
  signal (mutex)
end
```

Zasadnicze postulaty dotyczące warunkowych regionów krytycznych są spełnione, gdyż sprawdzanie warunku W odbywa się w obrębie regionu krytycznego, zaś oczekiwanie poza jego obrębem.

Procedury inicjowania i kończenia procesów można łatwo zbudować opierając się na buforze stanów początkowych statusów inicjowanych procesów. Nadawcami statusów do bufora są procesy inicjujące, zaś odbiorcami - procesy zakończone.

Zauważmy dodatkowo, że synchronizacja procesu programowanego z procesem działania urządzenia zewnętrznego również może być zrealizowana na bazie semaforów. Elegancki przykład rozwiązania problemu podaje Habermann w [5] (patrz Algorytm 2). Procedura deposit (d) ładuje do bufora urządzenia meldunek d i nadaje sygnał startu urządzenia (signal (message)) . Działanie urządzenia jest zaś opisane procedurą accept. Urządzenie po otrzymaniu sygnału startu przetwarza meldunek, po czym nadaje sygnał końca pracy (signal (frame)) . Istotą tego podejścia jest utożsamienie operacji zgłoszenia przerwania z operacją signal.

ALGORYTM 2
"SYNCHRONIZACJA PROCESU PROGRAMOWANEGO I ZEWNĘTRZNEGO

```
procedure deposit (d);
begin
   wait (free);
   buffer:=d;
   signal (message);
   wait (frame);
   signal (free);
end

procedure accept;
begin
   wait (message);
   process (buffer);
   signal (frame);
end
```

3. PRZYKŁAD REALIZACJI MONITORA

Wychodząc z przesłanek wymienionych powyżej, tzn. przyjmując jako minimalny, kompletny zbiór operacji monitora operacje wait i signal na semaforach, naszkicujemy poniżej ogólny zarys procedury obrazującej cykl rozkazowy procesora komputera wieloprocesorowego z uwypukleniem funkcji monitora.

Rozpatrujemy komputer składający się z u identycznych procesorów współpracujących ze wspólną pamięcią operacyjną.

W komputerze może być realizowanych współbieżnie p procesów programowanych. Procesy programowane synchronizują się wzajemnie i z procesami zewnętrznymi za pośrednictwem s semaforów. Każdy z procesów ma przypisany jeden z l poziomów priorytetowych, na podstawie którego monitor podejmuje decyzję o wywłaszczeniu procesu. Cykl rozkazowy każdego z procesorów przedstawia Algorytm 3. W algorytmie zastosowano zapożyczoną od Hansena [2] notację dla zmiennej q typu kolejka, zawierającej elementy typu P podzielone na l poziomów priorytetowych o postaci

```
type L = 1..l;
var q : queue L of P;
```

Procedury wprowadź (p, l'q) oraz usuń (p,l,q) wprowadzają i usuwają element p typu P z poziomu l kolejki q. Funkcja boolowska pilny (l, q) określa, czy kolejka q zawiera element, który jest ważniejszy niż inny element o danym poziomie l.

Do obsługi semaforów zadeklarowanych przez

```
semafor: array S of record
                    licznik: integer;
                    czekanie: queue L of P
                  end
```

wprowadzono funkcję boolowską zgłoszenie (semafor), która przyjmuje wartość "true", gdy macierz semafor zawiera co najmniej jeden element taki, że licznik ≠ 0 oraz kolejka czekanie nie jest pusta.

Procedura identyfikuj (przyczyna, semafor) wyznacza indeks (przyczyna) typu S elementu macierzy semafor, dla którego: licznik ≠ 0 oraz kolejka czekanie nie jest pusta.

Procedura wykonaj rozkaz (status (proces), wspólne, pamięć) wyznacza i wykonuje jedną z operacji przewidzianych w liście rozkazów procesora, z których interesują nas dwie związane z synchronizacją a opisane w Algorytmie 4.

Proces realizowany pod kontrolą przedstawionego monitora może znajdować się w jednym z trzech zasadniczych stanów:

działanie, gotowość, zawieszenie.

Wpisanie identyfikatora procesu do rejestru własnego procesora nazwanego "proces" (w wyniku wykonania procedury usuń (proces, poziom, gotowość)) powoduje przejście procesu w stan działania. Procesor wykonuje wtedy kolejno rozkazy wyznaczone przez status tego procesu.

Warunkiem koniecznym na to, aby proces był w stanie działania jest brak w kolejce gotowości procesu o poziomie ważniejszym niż rozpatrywany proces.

W przeciwnym przypadku proces przechodzi w stan gotowości, co jest związane z wprowadzeniem identyfikatora procesu do kolejki gotowości, Ważniejszy proces przechodzi wtedy w stan działania.

Wykonanie przez proces rozkazu zawierającego operację wait związane jest z przejściem procesu w stan zawieszenia, co polega na wprowadzeniu identyfikatora procesu do kolejki czekania semafora określonego przez parametr operacji.

ALGORYTM 3
"CYKL ROZKAZOWY PROCESORA FIZYCZNEGO"

```
type P= 1..p; L= 1..l; S= 1..s;
     T1= array  P  of status procesu;
     T2= record
          semafor: array  S  of record
                                licznik: integer;
                                czekanie: queue L of P
                                end;
          gotowość: queue L of P
          end
procedure cykl rozkazowy procesora (var status: T1; wspólne:
           shared T2; pamięć T);
var proces, nowy: P;
    poziom, pilność: L;
    przyczyna: S;
```

```
begin
   repeat
      region wspólne do with wspólne do
      begin
         while zgłoszenie (semafor) do
         begin
            identyfikuj (przyczyna, semafor);
            with semafor (przyczyna) do
            begin
               licznik:= licznik -1;
               usuń (nowy, pilność, czekanie)
            end;
            wprowadź (nowy, pilność, gotowość)
         end;
         if pilny (poziom, gotowość) then
         begin
            wprowadź (proces, poziom, gotowość);
            usuń (proces, poziom, gotowość)
         end;
      end;
      wykonaj rozkaz (status (proces), wspólne, pamięć);
   forever
end
```

ALGORYTM 4
"OPERACJE SYNCHRONIZUJĄCE PROCESORA"

```
procedure wait (s:S; var wspólne shared T2;
          proces: P; poziom: L);
begin
   region wspólne do with wspólne do
   begin
      with semafor (s) do
      wprowadź (proces, poziom, czekanie);
      usuń (proces, poziom, gotowość)
   end;
end
```

```
procedure signal (s:S; var wspólne: shared T2);
begin
   region wspólne do with wspólne do with semafor (s) do
   licznik:= licznik +1
end
```

Ze stanu zawieszenia proces przechodzi w stan gotowości po wybraniu go z kolejki czekanie w przypadku gdy procesor wykryje, że licznik semafora jest większy od zera. Wybranie jest związane ze zmniejszeniem licznika o 1. Zwiększenie licznika o 1 następuje w wyniku wykonania operacji signal przez proces programowany lub zewnętrzny.

4. PROCESOR WIRTUALNY DLA PROCESU PROGRAMOWANEGO

Przyjmując, że czas zajętości regionów krytycznych zmiennej "wspólne" jest pomijalnie mały w stosunku do czasu, w którym regiony te są wolne, tak że przy wejściu do tych regionów nie tworzą się kolejki, działanie komputera składającego się z u procesorów, w których działa opisany powyżej monitor równoważne jest działaniu maszyny wirtualnej złożonej z p procesorów wirtualnych z dużo prostszymi logicznie monitorami. Prostota logiczna osiągnięta została dzięki możliwości przydzielenia każdemu procesorowi programowanemu procesora wirtualnego, co dopuszcza realizację ich wzajemnej synchronizacji na bazie aktywnej formy czekania (busy form of waiting). Cykl rozkazowy procesora wirtualnego przedstawia Algorytm 5, natomiast operacje synchronizujące tego procesora - Algorytm 6. W Algorytmie 5 poza określonymi wcześniej procedurami zastosowano boolowską funkcję priorytet (proces, czekanie), która określa, czy element proces ma maksymalny priorytet. Jako element o maksymalnym priorytecie w kolejce traktuje się ten element, który zostałby wybrany z tej kolejki przez procedurę usuń (proces, poziom, czekanie). Ponieważ procedura usuń (proces, poziom, czekanie) występuje wraz z funkcją priorytet (proces, czekanie) w obrębie regionu krytycznego parametr "proces" procedury "cykl rozkazowy procesora" może być zdefiniowany jako stały.

ALGORYTM 5
"CYKL ROZKAZOWY PROCESORA WIRTUALNEGO"

```
const procesor = 0;
type P=1..p; L=1..l; S=1..s; VS= procesor.. s;
    VT= array  VS of record
                        licznik: integer;
                        czekanie: queue L of P
                     end;

procedure cykl rozkazowy v procesora (proces: P; var v semafor:
         shared VT: pamięć: T);
var   rejestry:  status procesu;
      poziom: L;
      stan: (działanie, gotowość, zawieszenie);
 procedure v wait (s: S; var v semafor: shared VT);
 var czekaj: boolean;
 begin
   czekaj:= true;
   region v semafor do with v semafor (s) do
   wstaw (proces, poziom, czekanie);
   repeat
     region v semafor do with v semafor (s) do
     if priorytet (proces, czekanie) and licznik > 0 then
     begin
       licznik:= licznik -1;
       usuń (proces, poziom, czekanie);
       czekaj:= false
     end;
   until not czekaj;
 end;

 procedure v signal (s: VS; var v semafor: shared VT);
 begin
   region v semafor do with v semafor (s) do
   licznik:= licznik +1
 end;
```

```
begin
  repeat
    if stan= gotowość then
    begin
      v wait (procesor, v semafor);
      stan:= działanie
    end;
    wykonaj rozkaz (rejestry, v semafor, pamięć);
    if stan = działanie then
    region v semafor do with v semafor (procesor) do
    if pilny (poziom, czekanie) then
    begin
      stan:= gotowość;
      v signal (procesor, v semafor)
    end;
  forever
end
```

ALGORYTM 6
"OPERACJE SYNCHRONIZUJĄCE PROCESORA WIRTUALNEGO"

```
procedure signal (s: S; var v semafor:  shared VT);
begin
  v signal (s, v semafor)
end

procedure  wait (s: S; var v semafor: shared VT,
                 stan: (działanie, gotowość, czekanie)) ;
begin
  stan:= czekanie;
  v signal (procesor, v semafor);
  v wait (s, v semafor);
  stan:= gotowość
end
```

5. PODSUMOWANIE

Jeśli monitor realizowany jest metodą układowo-programową, to jego część programowa ma charakter procesu programowanego. Zauważmy jednak istotne różnice we właściwościach monitora programowanego i procesu programowanego. Monitor programowany, jakkolwiek posiada własny status, współpracuje jednak z procesem sterującym realizującym jedynie funkcje sterowania sekwencyjnego. Ograniczenie funkcji procesu sterującego do potrzeb monitora programowanego osiąga się zazwyczaj metodą zablokowania przerwań. Z tego powodu programową realizację monitora można traktować jako swego rodzaju "wybieg" usuwający określone braki procesora.

Minimalizacja monitora poza względami natury ekonomicznej (minimalizacja sprzętu do realizacji funkcji synchronizacji) ma duże znaczenie również przy jego realizacji programowej. Pozwala bowiem tworzyć złożone narzędzia synchronizacji o budowie strukturalnej i osiągnąć to, co Dijkstra [6] nazywa możliwie najprostszą dolną warstwą oprogramowania.

Przedstawiony w Algorytmach 3 i 4 sposób obsługi semaforów zawiera pewną odmianę aktywnej formy czekania, polegającą na sprawdzaniu, czy oczekiwane zdarzenie zaszło. Zasadniczym powodem jej wprowadzenia była chęć uwypuklenia identycznej roli sygnałów zgłoszeń przerwań i sygnałów nadawanych rozkazem signal (S) dla synchronizacji procesów.

Ta metoda, aczkolwiek niemożliwa do przyjęcia w realizacji programowej ze względu na czasochłonność, wydaje się opłacalna przy realizacji układowej. Zauważmy, że jest ona powszechnie stosowana (i w wielu przypadkach jedynie możliwa) przy obsłudze części semaforów związanych z synchronizacją procesów programowych i zewnętrznych i zwana jest potocznie wykrywaniem przerwań. Rozciągnięcie jej na wszystkie semafory nie wydaje się być kłopotliwe.

Pojęcie procesora wirtualnego wprowadzone zostało tu przede wszystkim ze względów praktycznych, jako forma prostego, ale precyzyjnego opisu funkcji monitora z pominięciem opisu złożonego sposobu jego działania.

Literatura

[1] HORNING J.J.; RANDELL B.: Process Structuring, Technical Report CSRG, 1972, nr 15.

[2] BRINCH-HANSEN P.B.: Operating System Principles Prentice-Hall, 1973

[3] BRINCH-HANSEN P.B.: Structured Multiprogramming. CACM, 1972, t. 15, nr 7.

[4] WIRTH N.: The Programming Language "Pascal" (Revised Report), International Summer School on Structured Programming and Programmed Structures, Munich, Germany, Aug. 1973.

[5] HABERMANN A.N.: Synchronization of Communicating Processes, CACM, 1972, t. 15, nr 3.

[6] DIJKSTRA E.W.: Hierarchical Ordering of Sequential Processes, Acta Informatica, 1971, nr 1.

[7] DIJKSTRA E.W.: The Structure of the "THE" Multiprogramming System, CACM, 1968, t. 11, nr 5.

СИНХРОНИЗАЦИЯ ПРОЦЕССОВ В ВЫЧИСЛИТЕЛЬНОЙ МАШИНЕ

Резюме

На основе общего определения процесса вводятся понятия программированного процесса и монитора. Программированный процесс представляет собой комбинацию основных процессов, соответствующих командам программы. Монитор является синхронизирующим процессом для программированных и внешних процессов.

В труде рассматриваются два алгоритма командного цикла процессора. Первый описывает монитор с минимальным и комплектным множеством синхронизирующих функций. Второй - это алгоритм командного цикла виртуального процессора, в качестве собственного процессора для каждого программированного процесса.

SYNCHRONIZATION OF PROCESSES IN COMPUTER

Summary

On the base of general definition of process the concepts of a programmed process and monitor are introduced. A programmed process is a serial combination of basic processes corresponding to the instructions of the program. A monitor is a process synchronizing programmed and external processes.

Two algorithms of processor cycle are presented. The first one describes the monitor with minimal and complete set of synchronizing functions. The other one is the cycle algorithm of virtual processor as a private processor for each programmed process.

II. WIELOPROCESOROWOŚĆ A STRUKTURY OBLICZEŃ WSPÓŁBIEŻNYCH

ALGORYTMY, Zeszyt Specjalny, 1974, 71-84

681.322.06:
061.3/438/"1973"

PEWNE ASPEKTY WIELOPROCESOROWOŚCI

Andrzej ROWICKI
Instytut Maszyn Matematycznych

Pracę złożono 10.I.1974

W pracy dokonano pobieżnego przeglądu zagadnień związanych z realizacją wieloprocesorową obliczeń (programów). Między innymi omówiono aspekty językowe równoległości, przemienność i równoległość algorytmów oraz algorytmy o strukturze drzewa. Więcej uwagi poświęcono zagadnieniom bardziej ogólnym i trudniejszym pojęciowo. Zagadnienia natury bardziej technicznej nie zostały omówione.

Spis treści

1. UWAGI WSTĘPNE

W artykule dokonany zostanie przegląd zagadnień związanych z wieloprocesorową realizacją obliczeń (programów) w kolejności związanej z poziomem wymaganych informacji o wieloprocesorowych obliczeniach i procesach równoległych. Przegląd będzie zawierać aspekty językowe, a następnie zagadnienia przemien-

ności i równoległości algorytmów, które już wymagają pewnych informacji o systemie wieloprocesorowym związanych ze sposob komunikowania się z pamięcią systemu, dalej zagadnienia związane z wykrywaniem równoległości na poziomie wyrażeń arytmetycznych oraz zagadnienie określania czasu i liczby procesor dla dowolnych obliczeń, których struktura jest reprezentowana przez drzewo.

W początkowej fazie stosowania maszyn cyfrowych - mimo że były one znacznie szybsze od istniejących urządzeń mechanicznych analityczno-liczących - okazało się, że dla wielu zastosowań istniejące systemy są zbyt wolne. W związku z tym próbowano zwiększyć szybkość działania elementów podstawowych, z których zbudowana jest maszyna. Jednakże (mimo znacznych sukcesów) istnieją na tej drodze dość znaczne ograniczenia natury fizycznej i technologicznej. Dość wcześnie przekonano się, że szybkość działania maszyn można zwiększyć również poprzez zmianę struktury systemu (organizacja) i to przy mniejszym nakładzie środków. Zmiany organizacji systemu cyfrowego oprócz przyśpieszenia wykonywania poszczególnych operacji czy też zespołów operacji polegają również na tym, że umożliwiają równoczesne (równoległe) wykonywanie poszczególnych operacji czy też zespołów operacji.

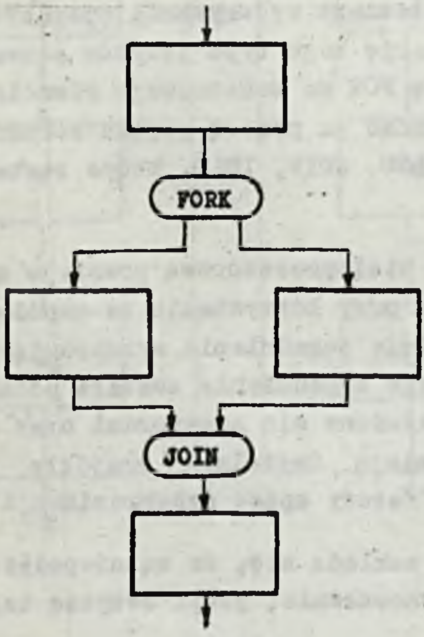
Przykładem, w którym uwzględniono "równoległość działania" są maszyny z podziałem czasu [7]. Wprawdzie trudno tu mówić o pełnej równoległości działania programów, poza tym na pierwszy rzut oka może się wydawać, że taki tryb pracy nie powinien dawać żadnych korzyści w sensie zwiększenia szybkości działania, to jednak w praktyce okazuje się, że taki tryb pracy niejednokrotnie znacznie zwiększa szybkość wykonywania zadań ze względu na to, że pewne fragmenty programów muszą oczekiwać na komunikację z urządzeniami zewnętrznymi. Rzeczywistą równoczesność działania programów uzyskuje się w maszynach wieloprocesorowych.

W związku z pojawieniem się maszyn wieloprocesorowych wystąpiło wiele problemów związanych ze strukturą systemu (architektura systemu), współpracą z pamięciami i urządzeniami peryfer

nymi, realizacją techniczną oraz związanych z realizacją programów (obliczeń). W dalszym ciągu omówimy nieco szerzej tylko grupę problemów i zagadnień związanych z realizacją wieloprocesorową procesów.

2. ASPEKTY JĘZYKOWE WIELOPROCESOROWOŚCI

Pierwszym typem zagadnień są sprawy związane z językami ułatwiającymi realizację wieloprocesorową programów. Jedną z pierwszych propozycji odnoszącą się do języków programowania [5] było wprowadzenie operacji FORK i JOIN podających informacje, że procesy (programy) zawarte między tymi operacjami mogą być wykonywane równocześnie. Ogólnie mówiąc operacja FORK inicjuje równoczesne wykonanie programu bezpośrednio po niej następującego oraz programu wskazanego przez jej parametr, a operacja JOIN inicjuje program wskazany przez jej parametr, o ile wszystkie programy zainicjowane przez operację FORK zostały zakończone. Powyższą sytuację można przedstawić schematycznie w sposób następujący:



Rys. 1

Jeżeli chodzi o języki wyższego poziomu, to jedną z pierwszych propozycji było wprowadzenie "nawiasów językowych" PARBEGIN i PAREND [6], które zastosowano do programów pisanych w ALGOL-u. Jest to metoda na tyle ogólna, że może być stosowana do różnych języków wyższego poziomu. Przykładem tego mogą być nawiasy COBEGIN i COEND [9], które zastosowano do programów pisanych w języku PASCAL [17]. Ogólnie mówiąc, wyrażenia zawarte między tego typu nawiasami mogą być realizowane w dowolnej kolejności, a więc w przypadku realizacji wieloprocesorowej - równocześnie. Różnica między tymi podejściami polega na tym, że operacje FORK i JOIN zawierają pewne informacje o sposobie realizacji równoczesnej programów, natomiast nawiasy językowe podają informacje o niezależności od siebie wyrażeń zawartych między tymi nawiasami.

Innym podejściem do zagadnienia jest tworzenie języków uwzględniających strukturę rozważanego systemu wieloprocesorowego. Przykładem takiego rozwiązania jest język algolopodobny TRANQUIL [1], zaprojektowany specjalnie dla maszyny ILLIAC IV.

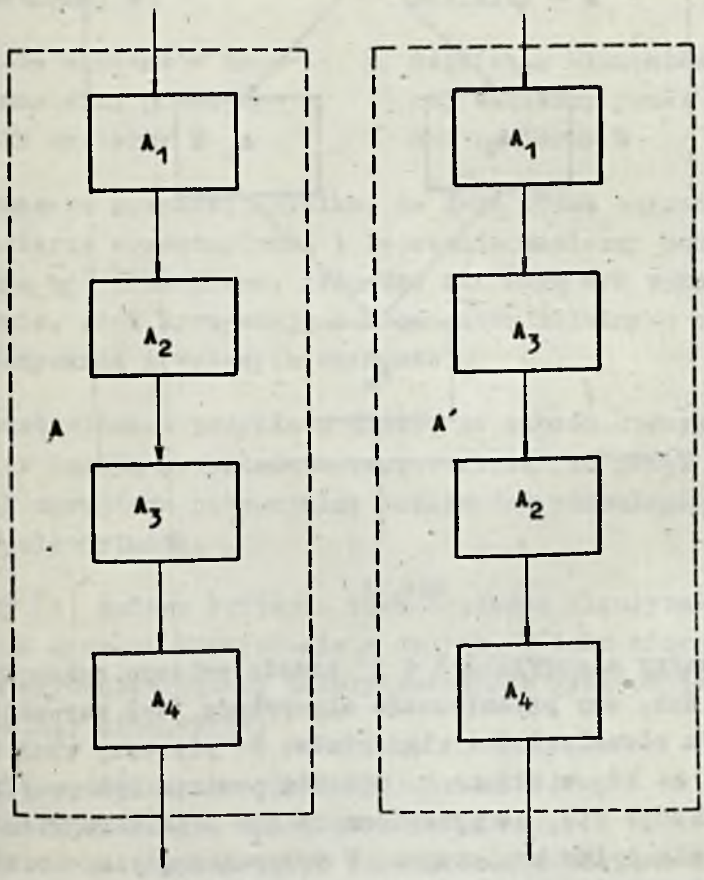
W językach algolopodobnych konstrukcja FOR zawiera potencjalne możliwości równoczesnego wykonywania operacji. W pracy [8] zaproponowano realizację tego typu języków pozwalającą przekształcić konstrukcję FOR na konstrukcję równoległą (PARALLEL FOR). Można tego dokonać za pomocą pięciu stosunkowo prostych funkcji PREP, AND, ALSO, JOIN, IDLE, które zostały szczegółowo omówione w [8].

W związku z pracą wieloprocesorową powstaje zagadnienie uniknięcia konfliktów przy korzystaniu ze wspólnych zasobów, inaczej mówiąc, powstaje zagadnienie synchronizacji procesów. Ogólne rozwiązanie tego zagadnienia zostało podane w [6]. W tym rozwiązaniu posłużono się semaforami oraz operacjami P i V, których definicję Czytelnik znajdzie w referacie T. Englerta pt. "Metody opisu synchronizacji procesów".

O operacjach tych zakłada się, że są niepodzielne i nie mogą być wykonywane równocześnie, jeśli dotyczą tego samego semafora.

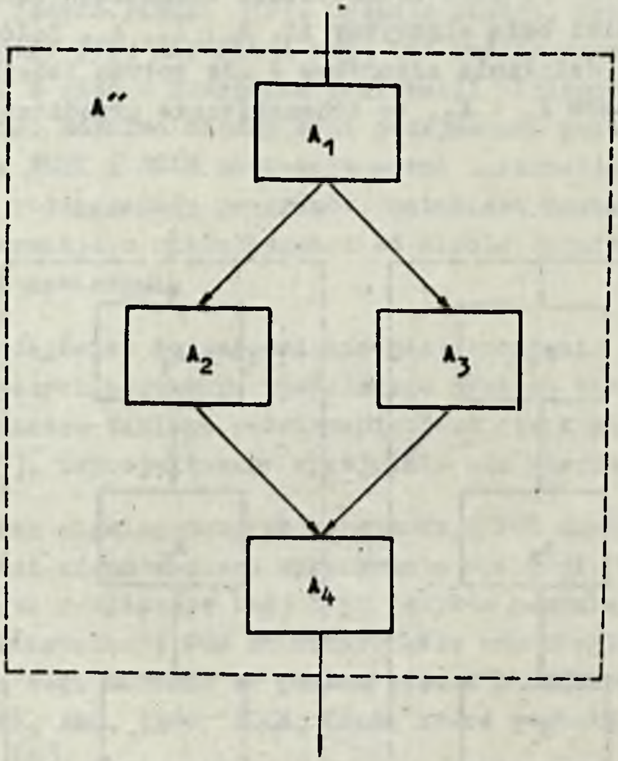
3. RÓWNOLEGŁOŚĆ I PRZEMIENNOŚĆ ALGORYTMÓW

Jednym z bardziej ogólnych zagadnień jest określenie warunków, kiedy algorytm realizujący dany proces w sposób sekwencyjny może być przekształcony w algorytm realizujący dany proces w sposób równoległy [4]. Rozważmy pewien algorytm A i załóżmy, że możemy w nim wyróżnić pewne części stanowiące zamkniętą całość. Niech nimi będą algorytmy A_1, A_2, A_3, A_4. Załóżmy, że na wynik końcowy działania algorytmu A nie wpływa kolejność realizacji algorytmów A_2 i A_3, co schematycznie przedstawiono na rysunku 2:



Rys. 2

Algorytmy A i A' są równoważne w sensie tych samych wyników końcowych. Jeżeli algorytmy A i A' są równoważne w sensie tych samych wyników końcowych, to algorytmy A_2 i A_3 nazywamy algorytmami przemiennymi. W związku z powyższym powstaje pytanie, czy algorytm A" przedstawiony schematycznie na rysunku 3



Rys. 3

jest równoważny algorytmom A i A' przedstawionym schematycznie na rys. 2, tzn. czy przemienność algorytmów jest warunkiem dostatecznym równoległości algorytmów. Na pierwszy rzut oka wydaje się, że odpowiedź na to pytanie powinna być pozytywna. Jednakże okazuje się, że przemienność nie jest warunkiem dostatecznym ale tylko koniecznym. W przypadku gdy zachodzi przemienność algorytmów oraz wartości argumentów algorytmów przemiennych zależą od kolejności ich realizowania, to wówczas nie można tych algorytmów realizować równocześnie (równolegle).

Przykładem ilustrującym niedostateczności przemienności jako warunku równoległości może być algorytm obliczania macierzy odwrotnej [12]. Przy obliczaniu macierzy odwrotnej do macierzy M można zastosować następujące sekwencje czynności:

I	II
1. utworzenie macierzy przestawionej - M^T	1. utworzenie macierzy podwyznaczników macierzy M - M′
2. utworzenie macierzy podwyznaczników macierzy przestawionej M^T	2. utworzenie macierzy przestawionej podwyznaczników macierzy - M'^T
3. dzielenie elementów uzyskanej macierzy przez wyznacznik macierzy M	3. dzielenie elementów uzyskanej macierzy przez wyznacznik macierzy M

Z powyższego przykładu wynika, że dwie różne czynności (tworzenie macierzy przestawionej i tworzenie macierzy podwyznaczników) mogą być zamienione, jednakże nie mogą być wykonywane równocześnie, gdyż korzystają z argumentów zależnych od kolejności wykonywania powyższych czynności.

Z przedstawionego przykładu widać, że sposób korzystania z danych (w naszym przykładzie reprezentowanych przez argumenty czynności) ogranicza potencjalne możliwości równoległości tkwiące w algorytmach.

W pracy [4] podano kryteria równoległości algorytmów w zależności od sposobu korzystania z danych. W celu sformułowania kryterium wyróżnimy cztery zbiory zmiennych występujących w algorytmie A_i, mianowicie:

1. W_i - zbiór zmiennych tylko pobieranych
2. X_i - zbiór zmiennych tylko pamiętanych
3. Y_i - zbiór zmiennych, które są najpierw pobierane a następnie pamiętane
4. Z_i - zbiór zmiennych, które są najpierw pamiętane a następnie pobierane.

Jeżeli przyjmiemy model maszyny, w którym procesory mogą się komunikować z pamięcią bezpośrednio, to warunek dostateczny wystąpienia równoległości dla algorytmów A_2 i A_3, spełniających założenia schematycznie przedstawione na rys. 2 i 3, jest następujący:

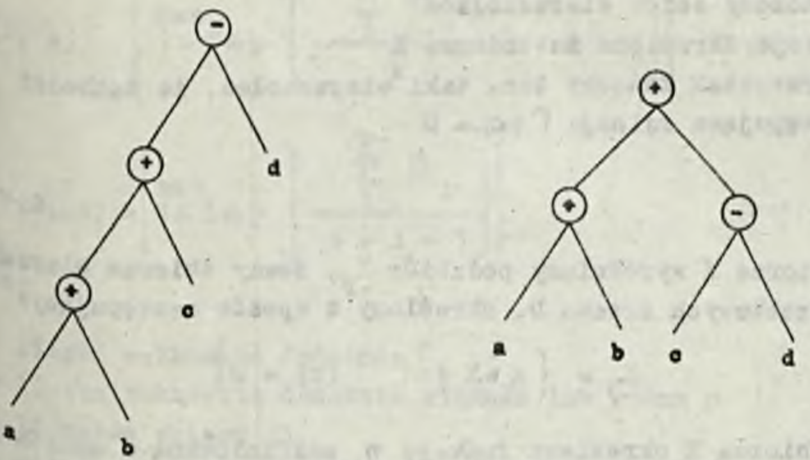
$$X_2 \cap X_3 \cap (W_4 \cup Y_4) = \emptyset$$

gdzie $\emptyset$ oznacza zbiór pusty.

Można również pokazać, że zagadnienie wykrywania przemienności i równoległości dowolnych algorytmów jest nierozstrzygalne [4], co oznacza że nie istnieje algorytm (ogólna metoda postępowania wspólna dla dowolnych algorytmów) pozwalający w skończonej liczbie kroków dać odpowiedź, czy rozważane algorytmy są przemienne czy też równoległe. Powyższe zagadnienie sprowadza się do problemu stopu dla maszyn Turinga, który jak wiadomo jest nierozstrzygalny.

4. ALGORYTMY O STRUKTURZE DRZEWA A RÓWNOLEGŁOŚĆ

Wykrywanie równoległości na poziomie wyrażeń arytmetycznych bazuje na znanym fakcie wzajemnej odpowiedniości wyrażeń arytmetycznych i drzew. Ogólnie mówiąc, zasada działania tych algorytmów [2, 10, 12, 15, 16] polega na "odtworzeniu" struktury drzewa, odpowiadającemu rozważanemu wyrażeniu arytmetycznemu, i na ustaleniu kolejności i równoczesności wykonywania poszczególnych operacji na podstawie odtworzonej struktury drzewa. Niektóre z nich bazując na prawie łączności operacji arytmetycznych [2, 15], zmieniają tak strukturę drzewa by uzyskać możliwie największą liczbę operacji wykonywanych równocześnie. Wyjaśnimy nieco bliżej tę sytuację: rozważmy przykładowo wyrażenie arytmetyczne (((a + b) + c) - d); korzystając z prawa łączności dodawania możemy omawiane wyrażenie przekształcić na następujące wyrażenie ((a+b) + (c-d)) . Drzewa odpowiadające tym wyrażeniom są następujące:



Rys. 4

Między omawianymi algorytmami zachodzą pewne różnice. Wynikają one z tych faktów, że poszczególne algorytmy są dostosowane do maszyn o różnej architekturze, do różnych języków, w których są opisane wyrażenia arytmetyczne, oraz są projektowane z punktu widzenia pewnych dodatkowych własności jak np.: zmniejszenia liczby "przeglądań" (analizowania) wyrażenia arytmetycznego, lub prostoty kodu generowanego przez algorytm.

W związku z realizacją wieloprocesorową obliczeń powstaje zagadnienie określenia czasu i liczby procesorów potrzebnych do wykonania danego obliczenia. W pracach [11, 13] podano ogólne rozwiązanie tego zagadnienia dla obliczeń, których struktura jest reprezentowana przez drzewo, a częściowe rozwiązanie dla obliczeń, których struktura zawiera "sklejone" fragmenty drzewa. Okazuje się, że w drugim przypadku można również uzyskać ogólne rozwiązanie tego zagadnienia [14]. Ze względu na konieczność wprowadzenia znacznej liczby dodatkowych pojęć ograniczymy się do dokładniejszego omówienia powyższego zagadnienia dla przypadku, gdy struktura obliczenia jest reprezentowana przez drzewo, a poszczególne etapy obliczenia są realizowane w jednakowym czasie.

Niech trójka uporządkowana $D = \langle X, \Gamma, \omega \rangle$ oznacza drzewo, gdzie

X - skończony zbiór wierzchołków
Γ - relacja określona na zbiorze X
ω - wierzchołek końcowy tzn. taki wierzchołek, że zachodzi następująca relacja $\Gamma(\omega) = \emptyset$

W zbiorze X wyróżniamy podzbiór X_p, zwany zbiorem wierzchołków początkowych drzewa D, określony w sposób następujący:

$$X_p = \left\{ x \in X : \Gamma^{-1}(x) = \emptyset \right\}$$

Na zbiorze X określamy funkcję η zdefiniowaną w sposób następujący:

$$\eta(x) = \begin{cases} 1 & \text{jeżeli } x = \omega \\ y + 1 & \text{jeżeli } \Gamma(x) = y \end{cases}$$

Liczbę p zdefiniowaną w sposób następujący

$$p = \max_{x \in X} \left\{ \eta(x) \right\}$$

nazywamy rzędem drzewa D.

Zbiór $\Omega_i \subset X$ określony w sposób następujący:

$$\Omega_i = \left\{ x \in X : \eta(x) = i \right\}$$

nazywamy i-tym piętrem drzewa D.

Po wprowadzeniu pojęć pomocniczych zdefiniujemy funkcje τ i $\varkappa$, przyporządkowujące liczby całkowite drzewu D w zależności od pewnych parametrów k i t, w sposób następujący:

$$\tau(D, k) = \left\lceil \max_{1 \le i \le p} \left\lceil \frac{\sum_{i}^{p} \bar{\bar{\Omega}}_j}{k} + i - 1 \right\rceil \right\rceil \qquad (*)$$

$$\varkappa(D, t) = \left\lceil \max_{1 \le i \le p} \left\lceil \frac{\sum_{i}^{p} \bar{\bar{\Omega}}_j}{t + i - 1} \right\rceil \right\rceil \qquad (**)$$

gdzie:

k - liczba całkowita dodatnia

t - liczba całkowita dodatnia większa lub równa p

$\bar{\bar{\Omega}}_i$ - liczność zbioru Ω_i

$\lceil x \rceil$ - najmniejsza liczba całkowita większa lub równa x

Algorytm wyznaczania wartości funkcji $\varkappa$ określony zależnością (* *) jest tylko nieznaczną modyfikacją algorytmu podanego w pracy [11], natomiast algorytm wyznaczania wartości funkcji τ określony zależnością (*) został podany w pracy [14].

Liczbom $\tau(D, k)$ i $\varkappa(D, t)$ można nadać określoną interpretację związaną z realizacją wieloprocesorową obliczenia.

Załóżmy, że mamy dane obliczenie P, które można rozłożyć na $P_1, P_2, \ldots, P_n$ segmentów rozłącznych o jednakowym czasie realizacji. Niech drzewo D reprezentuje strukturę obliczenia P w sposób następujący: między każdym wierzchołkiem drzewa D oraz segmentem obliczenia P istnieje wzajemnie jednoznaczne przyporządkowanie: jeżeli segment P_j ma być zrealizowany bezpośrednio po segmencie P_i, to między wierzchołkami x_j i x_i odpowiadającymi segmentom P_j i P_i zachodzi następująca relacja $x_j = \Gamma(x_i)$.

Jeżeli przyjmiemy taką interpretację drzewa D, to liczba $\tau(D, k)$ określa najmniejszą możliwą liczbę kroków (czas), która jest potrzebna do zrealizowania obliczenia P przy zadanej liczbie procesorów równej k. Natomiast liczba $\varkappa(D, t)$ określa najmniejszą możliwą liczbę procesorów potrzebną do zrealizowania obliczenia P w t krokach.

5. UWAGI KOŃCOWE

W pracy podano pobieżny przegląd pewnej części zagadnień występujących przy wieloprocesorowej realizacji programów (obliczeń). Przy opracowaniu referatu kierowano się zasadą by bardziej szczegółowo omówić zagadnienia natury bardziej ogólnej i zagadnienia trudniejsze pojęciowo. Pominięto wiele ważnych zagadnień występujących przy realizacji wieloprocesorowej programów, mających charakter bardziej techniczny i łatwiejszych pojęciowo. Szerszy zakres zagadnień związanych z realizacją wieloprocesorową programów można znaleźć w pracy [3]. Natomiast zagadnienie określania czasu i liczby procesorów dla obliczeń, których struktura jest reprezentowana przez twory bardziej złożone niż drzewa, zostały omówione w referacie autora pt. "Algorytmy określające liczbę procesorów oraz czas trwania obliczenia".

Literatura

[1] ABEL N.E. i inni: TRANQUIL: A Language for an Array Processing Computer, Proc. SJCC, 1969, t. 34, s. 57-73.

[2] BAER J.L., BOVET D.P.: Compilation of Arithmetic Expressions for Parallel Computations, Proc. of IFIP Congress 1968, t. 1, s. 340-346.

[3] BAER J.L.: A Survey of Some Theoretical Aspects of Multiprocessing, Computing Surveys, 1973, t. 5, nr 1, s. 31-80.

[4] BERNSTEIN A.J.: Analysis of Programs for Parallel Processing, IEEE Trans on EC, 1966, t. EC-15, nr 5, s. 751-763.

[5] CONWAY M.F.: A Multiprocessor System Design, Proc. FJ CC, 1963, t. 23, s. 139-146.

[6] DIJKSTRA E.W.: Co-operating Sequential Processes, Programming Languages. F. Genuys. Acad. Press. New York 1968, s. 43-112.

[7] GILL S.: Parallel Programming, The Computer Journal, 1958, t. 1, s. 2-10.

[8] GOSDEN J.A.: Explicit Parallel Processing Description and Control in Programs for Multi- and Uni-Processors Computers, Proc. FJCC, 1966, t. 29, s. 651-660.

[9] BRINCH-HANSEN P.: Operating System Principles, Prentice-Hall, 1973.

[10] HELLERMAN H.: Parallel Processing of Algebraic Expressions, IEEE Trans on EC, 1966, t. EC-15, nr 1, s. 82-91.

[11] HU T.C.: Parallel Sequencing and Assembly Line Problems, Operations Research, 1961, t. 9, nr 6, s. 841-848.

[12] RAMAMOORTHY C.V., GONZALES M.J.: A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs, Proc. FJCC, 1969, t. 35, s. 1-15.

[13] RAMAMOORTHY C.V., CHANDY K.M., GONZALES M.J.: Optimal Scheduling Strategies in a Multiprocessor Systems, IEEE Trans on Comp., 1972, t. C-21, nr 2, s. 137-146.

[14] ROWICKI A.: Związki między czasem realizacji obliczenia a liczbą procesorów (w przygotowaniu).

[15] SQUIRE J.S.: A Translation Algorithm for a Multiprocessor Computer, Proc. ACM 18th Nat. Conf. Denver 1963.

[16] STONE H.S.: One Pass Compilation of Arithmetic Expressions for a Parallel Processor, Comm. ACM, 1967, t. 10, nr 4, s. 220-223.

[17] WIRTH N.: The Programming Language Pascal, Acta Informatica, 1971, t. 1, s. 35-63.

НЕКОТОРЫЕ АСПЕКТЫ МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛЕНИЙ

Резюме

В труде дан общий обзор проблем, связанных с осуществлением многопроцессорных вычислений (программ) . В частности описаны языковые аспекты параллелизма, коммутативность и параллельность алгоритмов и алгоритмы, имеющие структуру дерева.

Особое внимание уделено более общим проблемам, связанным с введением понятий. Вопросы технического характера не затрагиваются.

SELECTED ASPECTS OF MULTIPROCESSING

Summary

This paper presents a brief survey of problems on multiprocessor realization of computations (programs).

There are considered, among others, lingual aspects of parallelism, commutativity and parallelism of algorithms and tree structure algorithms.

Attention is paid to general and notionally difficult questions rather than to problems of a technical nature.

ALGORYTMY, Zeszyt Specjalny, 1974, 85-100

681.322.06:
061.3/438/"1973"

ALGORYTMY OKREŚLAJĄCE LICZBĘ PROCESORÓW ORAZ CZAS TRWANIA OBLICZENIA

Andrzej ROWICKI
Instytut Maszyn Matematycznych

Pracę złożono 10.I.1974

W pracy podano dwa algorytmy pozwalające oszacować liczbę procesorów i czas trwania obliczenia. Powyższe algorytmy są poprawne dla obliczeń, które nie zawierają pętli oraz mają jedno wyjście.

W związku z realizacją wieloprocesorową obliczeń powstaje zagadnienie określenia czasu i liczby procesorów potrzebnych do wykonania danego obliczenia. Ten problem został częściowo rozwiązany w publikacjach [1, 2], mianowicie w pracy [1] podano algorytm wyznaczania liczby procesorów potrzebnych do wykonania obliczenia dla zadanego czasu (liczby kroków), w przypadku gdy struktura obliczenia jest reprezentowana przez drzewo. Natomiast w [2] podano algorytm wyznaczania liczby procesorów dla najkrótszego możliwego czasu (najmniejszej liczby kroków) realizacji obliczenia, w przypadku gdy struktura obliczenia jest reprezentowana przez dowolny graf nie zawierający pętli. W pracy przedstawimy dwa algorytmy pozwalające oszacować czas i liczbę procesorów potrzebnych do wykonania obliczenia nie zawierającego pętli, a mianowicie:

1. algorytm określający czas wykonania obliczenia przy z góry zadanej liczbie procesorów k,

2. algorytm określający liczbę procesorów potrzebną do wykonania obliczenia w z góry zadanym czasie t.

Powyższe algorytmy opiszemy dla modelu obliczenia reprezentowanego przez graf zorientowany. W celu uniknięcia niejasności przypomnimy podstawowe pojęcia z teorii grafów oraz zdefiniujemy pewne dodatkowe pojęcia potrzebne do opisu algorytmów.

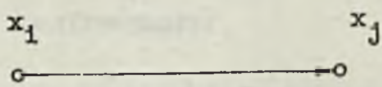
Rozważmy następujący graf $G = \langle X, \Gamma, \omega \rangle$, gdzie:

X – jest skończonym zbiorem wierzchołków

Γ – jest relacją zdefiniowaną na zbiorze X

ω – jest pewnym wyróżnionym wierzchołkiem zwanym wierzchołkiem końcowym grafu G, tzn. takim wierzchołkiem, że zachodzi następująca relacja $\Gamma(\omega) = \emptyset$ (gdzie $\emptyset$ oznacza zbiór pusty)

Jeżeli między wierzchołkami x_i i x_j zachodzi relacja $x_j \in \Gamma(x_i)$, to sytuację tę interpretujemy w sposób następujący



Rys. 1

Parę wierzchołków $u_i = \langle x_i, x_i' \rangle$ nazywamy łukiem, jeżeli zachodzą następujące relacje: $x_i' \in \Gamma(x_i)$ oraz $x_i \notin \Gamma(x_i')$. Natomiast parę wierzchołków $w_j = \langle x_j, x_j' \rangle$ nazywamy krawędzią, jeżeli zachodzą następujące relacje $x_j \in \Gamma(x_j)$ lub $x_j \in \Gamma(x_j')$.

Ciąg łuków $u_1, \ldots, u_i, \ldots, u_k$ nazywamy drogą, jeżeli zachodzą następujące relacje $x_1 \neq x_k'$ oraz dla każdego $1 \leqslant i < k$, $x_i' = x_{i+1}$. Jeżeli zamiast relacji $x_1 \neq x_k'$ zachodzi relacja $x_1 = x_k'$, to ciąg łuków $u_1, \ldots, u_i, \ldots, u_k$ nazywamy pętlą.

Ciąg krawędzi $w_1, \ldots, w_i, \ldots, w_k$ nazywamy łańcuchem, jeżeli zachodzi następująca relacja $x_i \neq x_k'$ oraz dla każdego $1 \leqslant i < k$, $x_i' = x_{i+1}$. Jeżeli zamiast relacji $x_i \neq x_k$ zachodzi relacja $x_i = x_k$, to ciąg krawędzi $w_1, \ldots, w_i, \ldots, w_k$ nazywamy obwodem.

Graf $G = \langle X, \Gamma, \omega \rangle$ nazywamy siecią, jeżeli

1. nie zawiera pętli
2. dla każdego wierzchołka $x \neq \omega$ istnieje droga do wierzchołka końcowego ω.

Natomiast sieć $G = \langle X, \Gamma, \omega \rangle$ będziemy nazywali drzewem, jeżeli nie zawiera ona obwodów.

W dalszych rozważaniach ograniczymy się tylko do sieci.

Zbiór $X_p \subset X$ zdefiniowany w sposób następujący

$$X_p = \left\{ x \in X: \quad \Gamma^{-1}(x) = \emptyset \right\}$$

będziemy nazywali zbiorem wierzchołków początkowych sieci G.

Natomiast zbiór $\Theta \subset X$ zdefiniowany w sposób następujący

$$\Theta = \left\{ x \in X: \overline{\overline{\Gamma(x)}} > 1 \right\}$$

będziemy nazywali zbiorem rozgałęzień sieci G, gdzie $\overline{\overline{X}}$ oznacza liczność zbioru X. Łatwo się przekonać, że jeżeli $\Theta = \emptyset$, to sieć G jest drzewem.

Na zbiorze X określimy dwie pomocnicze funkcje ξ i η zwane etykietami. Funkcje ξ i η definiujemy w sposób następujący

$$\xi(x) = \begin{cases} 1 \text{ jeżeli } x = \omega \\ \max\limits_{y \in (x)} \left\{ \xi(y) \right\} + 1 \text{ jeżeli } y \in \Gamma^{-1}(x) \end{cases}$$

Liczbę ρ zdefiniowaną w sposób następujący

$$\rho = \max_{x \in X} \left\{ \xi(x) \right\}$$

nazywamy rzędem sieci G.

$$\eta(x) = \rho + 1 - \xi(x)$$

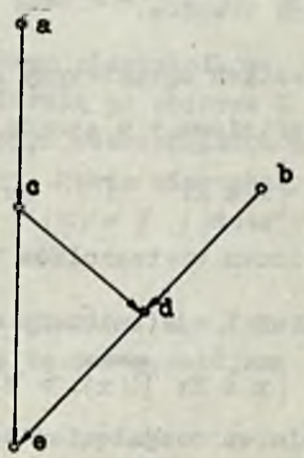
Łatwo zauważyć, że zachodzi następująca zależność

$$\rho = \max_{x \in X} \left\{ \eta(x) \right\}$$

Tytułem przykładu rozważmy sieć przedstawioną na rysunku 2.

Dla sieci przedstawionej na rys. 2 uzyskujemy

$$\omega = e$$
$$X_p = \{a, b\}$$
$$\Theta = \{c\}$$



Rys. 2

Natomiast wartości funkcji η i ξ są przedstawione w poniższej tabelce

Tabela 1

	a	b	c	d	e
ξ	4	3	3	2	1
η	1	2	2	3	4

Na mocy tabeli 1 uzyskujemy, że

$$\rho = 4$$

Na podstawie funkcji η zdefiniujmy teraz zbiór Ω_i w sposób następujący:

$$\Omega_i = \{x \in X: \eta(x) = i\}$$

Zbiór Ω_i nazywamy i-tym poziomem sieci G.

Do zdefiniowania funkcji φ i ψ, odgrywających zasadniczą rolę w naszych rozważaniach, będą potrzebne jeszcze pewne funkcje pomocnicze ν i ϑ. Funkcje ν i ϑ definiujemy w sposób następujący:

$$\nu(i) = \begin{cases} \text{największe takie } j < i, \text{ że } \Omega_j \cap \Theta \neq \emptyset \\ 0 \text{ w przeciwnym przypadku} \end{cases}$$

$$\vartheta(i, k) = \left\lceil \max_{\nu(i) < n \leqslant i} \left[\frac{\sum\limits_{\nu(i)+1}^{n} \overline{\overline{\Omega}}_j}{k} + i - n \right] \right\rceil$$

gdzie k jest liczbą całkowitą dodatnią, a $\lceil x \rceil$ oznacza najmniejszą liczbę całkowitą większą lub równą x.

Teraz przystąpimy do podania definicji funkcji φ i ψ.

Definicja φ

1° $\varphi(0, k) = 0$

2° $$\varphi(i+1, k) = \begin{cases} \varphi(i, k) + \overline{\overline{\Omega_{i+1}}} & \text{jeżeli } \Omega_{i+1} \cap \Theta = \emptyset \\ \varphi(\nu(i+1),k) + k \cdot \vartheta(i+1,k) & \text{jeżeli } \Omega_{i+1} \cap \Theta \neq \emptyset \end{cases}$$

W przypadku gdy $\Theta = \emptyset$, co oznacza, że rozpatrywana sieć jest drzewem, można pokazać, że

$$\varphi(i, k) = \sum_{1}^{i} \overline{\overline{\Omega}}_j$$

Definicja ψ

1° $$\psi(1,n,t) = \frac{\sum\limits_{1}^{\rho} \overline{\overline{\Omega}}_j}{t}$$

2° $$\psi(i+1,0,t) = \frac{\varphi(\rho - i, \lceil \psi(i, \mu(i), t) \rceil)}{t - i}$$

3° $$\psi(i+1, n+1, t) = \frac{\varphi(\rho - i, \lceil \psi(i+1, n, t) \rceil)}{t - i}$$

gdzie $\mu(i)$ oznacza najmniejsze takie n, że $\lceil \psi(i, n, t) \rceil = \lceil \psi(i, n+1, t) \rceil$

oraz t jest liczbą całkowitą dodatnią spełniającą następującą zależność $t \geqslant \rho$

mentem obliczenia P istnieje wzajemnie jednoznaczne przyporządkowanie; przyporządkowanie to jest takie, że jeżeli segment P_j ma być zrealizowany bezpośrednio po segmencie P_i, to między wierzchołkami x_j i x_i sieci G odpowiadającymi segmentom P_j i P_i obliczenia P zachodzi następująca relacja $x_j = \Gamma(x_i)$.

Można pokazać [3], że dla tak przyjętej interpretacji zachodzą następujące twierdzenia.

Twierdzenie 1

Niech T (G, k) oznacza liczbę kroków (jednostek czasu) potrzebną do zrealizowania obliczenia P przy zadanej liczbie procesorów równej k, a $\tau^*(G, k)$ funkcję określoną zależnością (*) to wówczas

$$\tau(G, k) \geqslant T(G, k) \geqslant \tau^*(G, k)$$

Twierdzenie 2

Niech G(n) oznacza sieć zawierającą n wierzchołków, to wówczas dla dowolnego n i dowolnego k istnieją takie sieci $G_i(n)$ i $G_j(n)$, że

$$T(G_i(n), k) = \tau(G_i(n), k) \quad \text{oraz}$$
$$T(G_j(n), k) = \tau^*(G_j(n), k)$$

Twierdzenie 3

Niech K (G, t) oznacza liczbę procesorów potrzebną do zrealizowania obliczenia P w czasie t (w t krokach), a $\varkappa^*(G, t)$ funkcję określoną zależnością (* *), to wówczas

$$\varkappa(G, t) \geqslant K(G, t) \geqslant \varkappa^*(G, t)$$

Twierdzenie 4

Niech $\rho(n)$ oznacza rząd sieci G(n), to wówczas dla dowolnego n i dowolnego $t \geqslant \rho(n)$ istnieją takie sieci $G_i(n)$ i $G_j(n)$, że

$$K(G_i(n), t) = \mathcal{K}(G_i(n), t) \quad \text{oraz}$$
$$K(G_j(n), t) = \mathcal{K}^*(G_j(n), t)$$

Z twierdzeń 1 i 3 oraz z zależności (*) i (* *) wynikają następujące twierdzenia.

<u>Twierdzenie 5</u>

Jeżeli sieć G jest drzewem, to

$$T(G, k) = \left\lceil \max_{1 \leqslant i \leqslant \rho} \left\{ \frac{\sum_{i}^{\rho+1-i} \Omega_j}{k} + i - 1 \right\} \right\rceil$$

oraz liczba T (G, k) jest najmniejszą możliwą liczbą kroków (jednostek czasu) potrzebną do zrealizowania obliczenia P przy zadanej liczbie procesorów równej k.

<u>Twierdzenie 6</u>

Jeżeli sieć G jest drzewem, to

$$K(G, t) = \left\lceil \max_{1 \leqslant i \leqslant \rho} \left\{ \frac{\sum_{1}^{\rho+1-i} \bar{\Omega}_j}{t + 1 - i} \right\} \right\rceil^{*)}$$

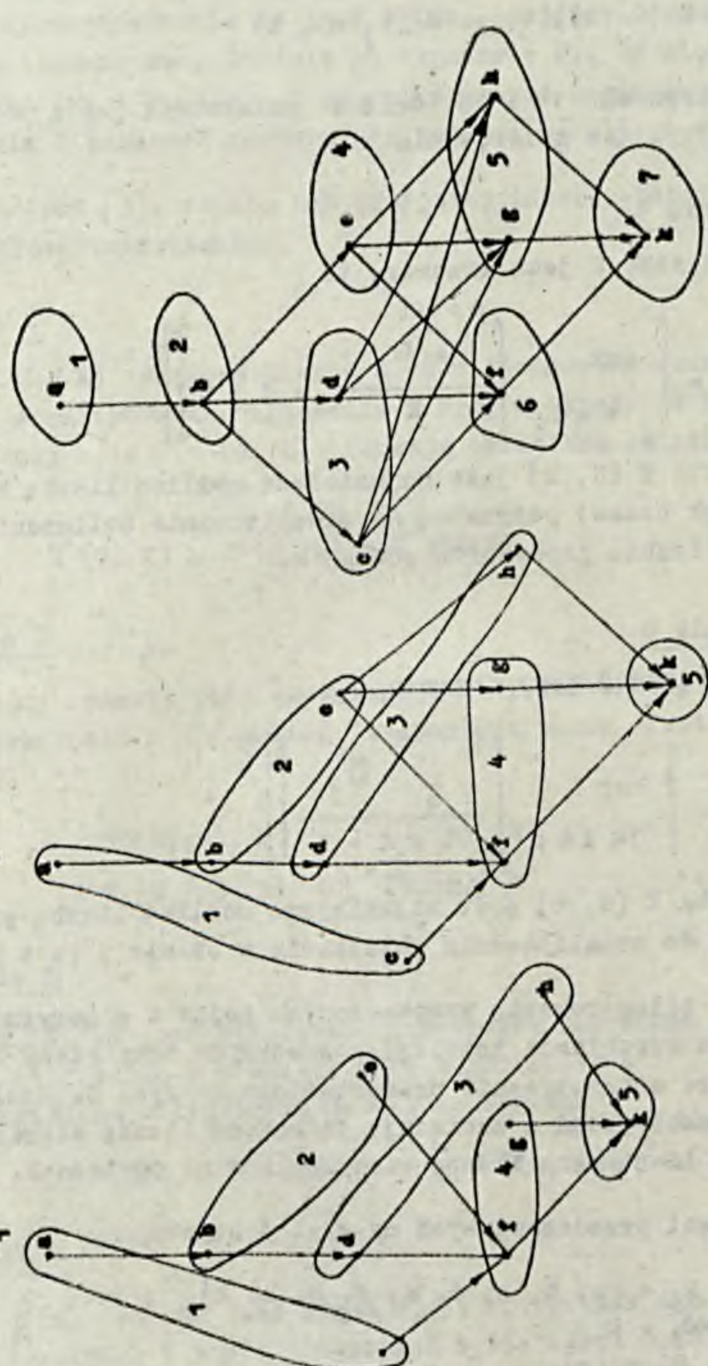
oraz liczba K (G, t) jest najmniejszą możliwą liczbą procesorów potrzebną do zrealizowania obliczenia w czasie t (w t krokach).

W celu zilustrowania wprowadzonych pojęć i algorytmów oraz ułatwienia wyrobienia intuicji rozpatrzymy trzy sieci G_1, G_2 i G_3, które schematycznie przedstawiono na rys. 3. Sieci te mają jednakowy rząd i zawierają jednakową liczbę wierzchołków oraz mają identyczną liczbę wierzchołków na poziomach.

Dla sieci przedstawionych na rys. 3 uzyskujemy

$x_1 = x_2 = x_3 = \{a, b, c, d, e, f, g, h, k\}$
$\omega_1 = \omega_2 = \omega_3 = k$
$\theta_1 = \emptyset \qquad \theta_2 = \{e\} \qquad \theta_3 = \{b, c, d, e\}$

*) Algorytm ten jest nieznaną modyfikacją algorytmu podanego w [1].



Rys. 3

Wartości funkcji ξ i η dla sieci G_1, G_2, G_3 są identyczne i podane w następującej tabelce:

Tabela 2

	a	b	c	d	e	f	g	h	k
$\xi_{1,2,3}$	5	4	3	3	3	2	2	2	1
$\eta_{1,2,3}$	1	2	3	3	3	4	4	4	5

Na mocy tabeli 2 uzyskujemy, że

$\rho = 5$ oraz

$\Omega_1 = \{a\}$

$\Omega_2 = \{b\}$

$\Omega_3 = \{c, d, e\}$

$\Omega_4 = \{f, g, h\}$

$\Omega_5 = \{k\}$

Załóżmy, że do dyspozycji mamy dwa procesory, tzn. że parametr $k = 2$. Dla tej wartości parametru wyznaczamy wartość funkcji φ dla sieci G_3.

Na mocy definicji funkcji φ uzyskujemy

$$\varphi_3\ (0,2) = 0$$

ponieważ $\Omega_1 \cap \Theta = \emptyset$ to

$$\varphi_3\ (1,2) = \varphi_3(0,2) + \bar{\Omega}_1 = 1$$

ponieważ $\Omega_2 \cap \Theta \neq \emptyset$ to

$$\varphi_3\ (2,2) = \varphi_3(\vartheta(2),\ 2) + 2 \cdot \vartheta(2,2)$$

na mocy definicji funkcji ϑ uzyskujemy $\vartheta(2) = 0$, a więc

$$\vartheta(2,2) = \left\lceil \max_{0<n\leqslant 2} \left\{ \frac{\sum\limits_{n}^{2} \bar{\Omega}_j}{2} + i - n \right\} \right\rceil = \left\lceil \max\left\{1,\ 1{,}5\right\} \right\rceil = 2$$

Ponieważ $\varphi_3(\vartheta(1),2) = 0$, to uzyskujemy ostatecznie

$$\varphi_3(2,2) = 4$$

Ponieważ $\Omega_3 \cap \Theta \neq \emptyset$, to

$$\varphi_3(3,2) = \varphi_3(\vartheta(3),k) + 2 \cdot \vartheta(3,2)$$

Ponieważ $\vartheta(3) = 2$, to

$$\varphi_3(\vartheta(3),k) = 4$$

oraz

$$\vartheta(3,2) = \left\lceil \max_{2<n\leq 3} \left\{ \frac{\sum_{n}^{3} \bar{\Omega}_j}{2} + i - n \right\} \right\rceil = 2$$

a więc uzyskujemy ostatecznie

$$\varphi_3(3,2) = 8$$

Ponieważ $\Omega_4 \cap \Theta = \emptyset$ oraz $\Omega_5 \cap \Theta = \emptyset$ to

$$\varphi_3(4,2) = \varphi_3(3,2) + \bar{\Omega}_4 = 11$$

$$\varphi_3(5,2) = \varphi_3(4,2) + \bar{\Omega}_5 = 12$$

Postępując w podobny sposób wyznaczamy wartości funkcji dla sieci G_1 i G_2. Dla $k = 2$ uzyskane wyniki zestawiono w poniższej tabelce

Tabela 3

i	1	2	3	4	5
φ_1	1	2	5	8	9
φ_2	1	2	6	9	10
φ_3	1	4	8	11	12

Teraz przystąpimy do wyznaczania liczby kroków potrzebnych do zrealizowania obliczeń, których struktura jest reprezentowana przez sieci G_1, G_2 i G_3.

Ponieważ sieć G_1 jest drzewem ($\Theta_1 = \emptyset$), to na mocy twierdzenia 5 i tabelki 3 uzyskujemy:

$$T(G_1, 2) = \left\lceil \max \left\{ 4, \frac{9}{2}, 5 \right\} \right\rceil = 5$$

Natomiast na mocy tabelki 3 funkcja τ dla sieci G_2 i G_3 ma następujące wartości:

$$\tau(G_2, 2) = \left\lceil \max \left\{ 4, \frac{9}{2}, 5, \frac{11}{2} \right\} \right\rceil = 6$$

$$\tau(G_3, 2) = \left\lceil \max \left\{ \frac{9}{2}, 5, 6, \frac{13}{2} \right\} \right\rceil = 7$$

Na mocy twierdzenia 1 uzyskujemy ostatecznie

$$6 \geqslant T(G_2, 2) \geqslant 5$$

$$7 \geqslant T(G_3, 2) \geqslant 5$$

Na rysunku 3 pokazano, że rzeczywiście można zrealizować obliczenia, których struktura jest reprezentowana przez sieci G_1, G_2, G_3 w liczbie kroków określonej rozważanymi algorytmami, oraz łatwo się przekonać na podstawie rys. 3, że nie można zrealizować obliczeń w mniejszej liczbie kroków. Najmniejsze możliwe liczby kroków dla sieci G_1, G_2, G_3 są następujące: 5, 5, 7.

Rozważymy teraz zagadnienie odwrotne do poprzedniego, a mianowicie dla zadanego czasu realizacji (liczby kroków) obliczenia wyznaczymy potrzebną liczbę procesorów. Wyliczeń dokonamy tylko dla sieci G_3 i dla następującej liczby kroków: 5, 6, 7.

Dla $t = 5$ na mocy definicji funkcji Ψ uzyskujemy:

$$\psi(1, n, 5) = \frac{9}{5} \quad \text{oraz}$$

$$\psi(1, \mu(1), 5) = \frac{9}{5}$$

Przystąpimy teraz do wyznaczania wartości funkcji ψ dla $i = 2$

$$\psi(2,0,5) = \frac{\varphi(4,\lceil\frac{9}{5}\rceil)}{4} = \frac{\varphi(4,2)}{4}$$

Na mocy rozważań z poprzedniego przypadku uzyskujemy, że $\varphi(4,2) = 11$, a więc

$$\psi(2,0,5) = \frac{11}{4}$$

$$\psi(2,1,5) = \frac{\varphi(4,\lceil\frac{11}{4}\rceil)}{4} = \frac{\varphi(4,3)}{4}$$

Przystąpimy teraz do wyznaczenia wartości funkcji φ dla $i = 4$ i $k = 3$.

Ponieważ $\Omega_4 \cap \Theta = \emptyset$ to

$$\varphi(4,3) = \varphi(3,3) + \bar{\Omega}_4 = \varphi(3,3) + 3$$

Ponieważ $\Omega_3 \cap \Theta \neq \emptyset$ oraz $\vartheta(3) = 2$, to

$$\varphi(3,3) = \varphi(2,3) + 3 \cdot \vartheta(3,3) \quad \text{oraz}$$

$$\vartheta(3,3) = \left\lceil \max_{2<n\leq 3} \left\{ \frac{\sum_{3}^{n} \bar{\Omega}_j}{3} + 3 - n \right\} \right\rceil = 1$$

stąd

$$\varphi(3,3) = \varphi(2,3) + 3$$

Ponieważ $\Omega_2 \cap \Theta \neq \emptyset$ oraz $\vartheta(2) = 0$, to

$$\varphi(2,3) = \varphi(0,3) + 3 \cdot \vartheta(2,3) \quad \text{oraz}$$

$$\vartheta(2,3) = \left\lceil \max_{0<n\leq 2} \left\{ \frac{\sum_{1}^{n} \bar{\Omega}_j}{3} + 2 - n \right\} \right\rceil =$$

$$= \left\lceil \max \left\{ \frac{2}{3}, \frac{4}{3} \right\} \right\rceil = 2$$

a więc

$$\varphi(2,3) = 6$$

Na podstawie przeprowadzonych obliczeń uzyskujemy ostatecznie

$\varphi(4,3) = 12$ a więc

$$\psi(2,1,5) = 3$$

Ponieważ $\psi(2,0,5) = \frac{11}{4}$ oraz $\psi(2,1,5) = 3$ to

$$\psi(2, \mu(2), 5) = \frac{11}{4}$$

Wyznaczymy teraz wartość funkcji ψ dla $i = 3$

$$\psi(3,0,5) = \frac{\varphi(3, \lceil \frac{11}{4} \rceil)}{3} = \frac{\varphi(3,3)}{3}$$

Na mocy poprzednich rozważań uzyskujemy

$\varphi(3,3) = 9$ a więc

$$\psi(3,0,5) = 3$$

$$\psi(3,1,5) = \frac{\varphi(3, \lceil 3 \rceil)}{3} = 3$$

co daje ostatecznie

$$\psi(3, \mu(3), 5) = 3$$

$$\psi(4,0,5) = \frac{\varphi(2, \lceil 3 \rceil)}{2}$$

Na mocy poprzednich rozważań $\varphi(2,3) = 6$ a więc

$$\psi(4,0,5) = 3$$

$$\psi(4,1,5) = \frac{\varphi(2, \lceil 3 \rceil)}{2} = 3$$

co daje ostatecznie

$\psi(4, \mu(4), 5) = 3$

$\psi(5,0,5) = \varphi(1, \lceil 3 \rceil)$

Ponieważ $\Omega_1 \cap \Theta = \emptyset$ to $\varphi(1, 3) = 1$ a więc

$\psi(5,0,5) = 1$

$\psi(5,1,5) = \varphi(1, \lceil 1 \rceil)$

Ponieważ $\varphi(1,1) = 1$ to

$\psi(5, \mu(5), 5) = 1$

Postępując podobnie jak dla $t = 5$ wyznaczamy wartości funkcji ψ dla $t = 6$ i $t = 7$. Uzyskane wyniki są podane w poniższej tabelce

Tabela 4

i	1	2	3	4	5
$\psi(i, \mu(i), 5)$	$\frac{9}{5}$	$\frac{11}{4}$	3	3	1
$\psi(i, \mu(i), 6)$	$\frac{9}{6}$	$\frac{11}{5}$	$\frac{9}{4}$	2	$\frac{1}{2}$
$\psi(i, \mu(i), 7)$	$\frac{9}{7}$	$\frac{11}{6}$	$\frac{8}{5}$	1	$\frac{1}{3}$

Na mocy definicji funkcji $\varkappa$ oraz tabeli 4 uzyskujemy:

$$\varkappa(G_3, 5) = \left\lceil \max \left\{1, \frac{9}{5}, \frac{11}{4}, 3\right\}\right\rceil = 3$$

$$\varkappa(G_3, 6) = \left\lceil \max \left\{\frac{1}{2}, \frac{9}{6}, \frac{11}{5}, 2, \frac{9}{4}\right\}\right\rceil = 3$$

$$\varkappa(G_3, 7) = \left\lceil \max \left\{\frac{1}{3}, 1, \frac{9}{7}, \frac{8}{5}, \frac{11}{6}\right\}\right\rceil = 2$$

Natomiast wartość funkcji $\varkappa^*$ wyznaczamy na podstawie zależności (* *)

$$\varkappa^*(G_3, 5) = \left\lceil \max \left\{1, \frac{5}{3}, \frac{9}{5}, 2\right\}\right\rceil = 2$$

$$\varkappa^*(G_3, 6) = \left\lceil \max \left\{ \frac{1}{2}, \frac{2}{3}, \frac{5}{4}, \frac{9}{6}, \frac{8}{5} \right\} \right\rceil = 2$$

$$\varkappa^*(G_3, 7) = \left\lceil \max \left\{ \frac{1}{3}, \frac{1}{2}, 1, \frac{9}{7}, \frac{8}{6} \right\} \right\rceil = 2$$

Na mocy twierdzenia 3 oraz na podstawie uzyskanych wyników otrzymujemy ostatecznie:

$$3 \geqslant K\ (G_3, 5) \geqslant 2$$

$$3 \geqslant K\ (G_3, 6) \geqslant 2$$

$$2 \geqslant K\ (G_3, 7) \geqslant 2$$

Na podstawie rysunku 3 można się przekonać, że obliczenia, którego struktura jest reprezentowana przez sieć G_3, nie można przy liczbie procesorów mniejszej od 3 zrealizować w 5 lub 6 krokach, natomiast dwoma procesorami można go zrealizować w liczbie kroków nie mniejszej niż 7, co jest zgodne z wynikiem uzyskanym na podstawie twierdzenia 3 oraz niesprzeczne z wynikiem uzyskanym na podstawie twierdzenia 1.

Literatura

[1] HU T.C.: Parallel Sequencing and Assembly Line Problems Operations Research, 1961, vol. 9, nr 6, s. 841-848.

[2] RAMAMOORTHY C.V., CHANDY K.M., GONZALES M.J.: Optimal Scheduling Strategies in a Multiprocessor Systems, IEEE Trans. on Comp., 1972, vol. C-21, nr 2, s. 137-146.

[3] ROWICKI A.: Związki między czasem realizacji obliczenia a liczbą procesorów (w przygotowaniu).

АЛГОРИТМЫ, ОПРЕДЕЛЯЮЩИЕ ЧИСЛО ПРОЦЕССОРОВ И ВРЕМЯ ВЫЧИСЛЕНИЯ

Резюме

В разработке представлены два алгоритма, позволяющие определить число процессоров и время вычисления. Представленные алгоритмы правильны для вычислений с одним выходом и не содержащих петлей.

ALGORITHMS ESTIMATING NUMBER OF PROCESSORS AND DURATION OF COMPUTATION

Summary

This paper presents two algorithms for the estimation of the number of processors and for the estimation of the computation duration. The algorithms are proven for computations containing no circuits and only one output.

III. MODELOWANIE W PROJEKTOWANIU I BADANIU SYSTEMÓW CYFROWYCH

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

ALGORYTMY, Zeszyt Specjalny, 1974, 103-120

681.322.06:
061.3/438/"1973"

MODELOWANIE I EKSPERYMENT SYMULACYJNY W PROJEKTOWANIU SYSTEMÓW CYFROWYCH

Piotr BIELKOWICZ
Piotr PERKOWSKI

Instytut Maszyn Matematycznych

Pracę złożono 10.I.1974

Wzrost złożoności współczesnych systemów cyfrowych, wynikający z rozwoju możliwości funkcjonalnych sprzętu, wysokie wymagania użytkowe dotyczące sprawności oprogramowania, zmuszają projektantów do szerokiego korzystania i rozwijania narzędzi programowych umożliwiających szybką weryfikację i ocenę ilościową koncepcji projektowych. Artykuł porusza istotne problemy, które wyłaniają się w czasie projektowania złożonych systemów za pomocą modelowania symulacyjnego. Omawiane problemy poparto przykładami praktycznymi.

Spis treści

WSTĘP

W niniejszej pracy rozpatrujemy problemy związane z wykorzy-staniem techniki symulacyjnej w projektowaniu i optymalizacji złożonych systemów cyfrowych. Mówiąc o optymalizacji (minimali-zacji lub maksymalizacji wybranych parametrów systemu dla sta-nów ustalonych) przyjmujemy, że wymagania użytkowe oraz kryte-ria oceny charakterystyk użytkowych projektowanego systemu zostały uprzednio sformułowane.

Możliwe są dwa podejścia do problemu projektowania systemu:

1. poszukiwanie rozwiązania dopuszczalnego, tj. spełniającego zbiór wymagań (ograniczeń),

2. poszukiwanie rozwiązania optymalnego, tj. najlepszego, w sen-sie przyjętego kryterium oceny, spośród rozwiązań dopuszczal-nych (np. maksymalna przepustowość systemu przekazywania me-dunków, minimalny czas odpowiedzi systemu wyszukiwania in-formacji, maksymalna liczba urządzeń końcowych w systemie wielodostępnym).

Wymienione wyżej problemy, rozpatrywane w fazie projektowa-systemu, wymagają posługiwania się modelem zastępującym rzeczy-wisty obiekt i umożliwiającym oszacowanie charakterystyk użytko-wych systemu. Najbardziej pożądanym typem modelu jest zawsze pełny model analityczny. Zwykle jednak, w przypadku systemów o złożonej, niejednorodnej strukturze, dużej liczbie zmiennych czy nieliniowych zależnościach między zmiennymi konstrukcja praktycznie użytecznego modelu analitycznego jest niemożliwa lub nieopłacalna. W tej sytuacji jedynym dostępnym, ekonomicz-uzasadnionym postępowaniem jest budowa modeli symulacyjnych o-eksperymentowanie z tymi modelami.

Podstawowymi problemami pojawiającymi się na etapie budowy modelu są weryfikacja poprawności modelu i minimalizacja model-

Może się zdarzyć, że w trakcie budowy modelu pominięte zost-ną zmienne i relacje między tymi zmiennymi istotne dla modelow-nego systemu. Możemy również popełnić błędy w modelowaniu stru-

tury systemu. W takim wypadku model nie jest poprawny, tzn. zachowanie się modelu dla danego strumienia danych wejściowych odbiega znacznie od zachowania się rzeczywistego systemu. Oczywiście nie można oczekiwać, że zachowanie się modelu będzie identyczne jak obiektu rzeczywistego. Rozbieżność powinna jednak mieścić się w racjonalnych granicach. Niepoprawny model jest oczywiście bezużyteczny. Jednym ze sposobów "polepszenia" modelu jest zwiększenie jego szczegółowości. Wiąże się to jednak ze wzrostem kosztów symulacji i analizy wyników.

Zwiększanie szczegółowości modelu kłóci się z dążeniem do minimalizacji modelu. Minimalizacja postuluje uwzględnienie tylko najistotniejszych zmiennych i relacji.

Problemom poprawności i minimalizacji modelu poświęcone są paragrafy 1 i 5. Mając model systemu i przystępując do eksperymentów na maszynie cyfrowej, należy wiele uwagi poświęcić właściwemu ich zaprojektowaniu. Celem każdego eksperymentu jest pogłębienie wiedzy o badanym systemie, w szczególności o związkach między zmiennymi uwzględnionymi w modelu, zależnościach między odpowiedzią (kryterium oceny) systemu a zmiennymi kontrolowanymi itp. Informacje te należy zdobyć minimalnymi kosztami, dlatego też projekt eksperymentu powinien uwzględniać problematykę zbieżności stochastycznej, doboru długości próbki itp.

Podstawowym problemom projektowania eksperymentu symulacyjnego poświęcony jest paragraf 2. W paragrafie 3 omawia się zbieżność stochastyczną i związane z nią problemy weryfikacji wyników symulacji i doboru długości próbki.

W paragrafie 4 omówione są dwa przykładowe problemy, które w dosyć prosty sposób mogą być rozwiązane za pomocą modelowania symulacyjnego. W paragrafie 5 podkreślono najistotniejsze sprawy związane z symulowaniem powyższych przykładów.

1. POPRAWNOŚĆ MODELU

Weryfikacja poprawności modelu jest jednym z najistotniejszych problemów z jakimi spotykamy się w naszych rozważaniach.

Wiąże się ona z wieloma zagadnieniami natury praktycznej, teoretycznej, statystycznej i nawet filozoficznej.

Najłatwiej jest weryfikować model w przypadku, gdy wyniki otrzymane z symulacji mogą być bezpośrednio porównywane z danymi pochodzącymi z systemu rzeczywistego. Jeżeli rozbieżności nie mieszczą się w dopuszczalnych granicach, model jest modyfikowany. Sytuacja komplikuje się, jeżeli nie mamy możliwości oparcia się na obiekcie rzeczywistym, tzn. wtedy, gdy system rzeczywisty jest dopiero projektowany. W przypadku, gdy system projektowany jest zbliżony do systemów istniejących, poprawność modelu możemy stwierdzić porównując go z tymi systemami. Należy jednak zachować ostrożność przy wyciąganiu wniosków, pamiętając o różnicach między systemem projektowanym a istniejącymi.

Sytuacja pogarsza się jeżeli dla projektowanego systemu trudno znaleźć analogiczny przypadek.

Proponuje się rozbicie modelu na podmodele o różnych stopniach szczegółowości. Na przykład model z grubsza opisujący działanie całego systemu może być prosty i zawierać mało zmiennych i relacji. Z drugiej strony model użyty do analizy algorytmu przydziału wybranej jednostki pamięci dyskowej powinien być bardziej szczegółowy.

Rozbicie powinno iść w tym kierunku, aby sprawdzenie poprawności "najniższych" podmodeli było już proste (np. podmodel przydziału pewnego urządzenia można porównywać ze znanymi modelami analitycznymi lub analogicznymi rozwiązaniami dla innych istniejących systemów). Oczywiście z poprawności podmodeli nie wynika poprawność całego modelu. Wnioskowanie o poprawności całego modelu będzie opierać się przede wszystkim na intuicji projektanta. Powyższe zagadnienia omówione są szerzej w [5] oraz w [9] i [10].

2. PROBLEMY ZWIĄZANE Z PROJEKTOWANIEM EKSPERYMENTU SYMULACYJNEGO

Należy zauważyć, że model symulacyjny jak każdy model nie powinien być rozpatrywany w oderwaniu od celów, którym służy

- sam model nie uzupełniony odpowiednim projektem eksperymentu może być rozpatrywany jedynie jako narzędzie opisu rzeczywistego obiektu. Upraszczając zagadnienie można przyjąć, że podstawowe problemy projektowania eksperymentu związane są z poszukiwaniem odpowiedzi na następujące pytania [4]:

1. Czy dany czynnik (zmienna niezależna, wejściowa) jest kontrolowalny?
   Czynnik uważamy za kontrolowalny wtedy, jeżeli jego wartości mogą być ustalone w czasie eksperymentu w sposób zamierzony, np. zgodny z określoną regułą decyzyjną.

2. Czy dany czynnik jest obserwowalny?
   Czynnik uważamy za obserwowalny jeśli jego wartości mogą być obserwowane lub mierzone i rejestrowane jako część danych. W eksperymentach symulacyjnych z natury rzeczy mamy do czynienia z czynnikami kontrolowalnymi i obserwowalnymi.

3. Czy dany czynnik jest czynnikiem podstawowym czy też pomocniczym zwiększającym jedynie dokładność eksperymentu?

4. Czy dany czynnik ma charakter ilościowy czy też jakościowy?
   Przykładem czynnika o charakterze jakościowym może być np. zmienna decyzyjna, wyrażająca pewną regułę decyzyjną.

5. Czy dany czynnik ma charakter przypadkowy?

Niezmiernie istotnym problemem z punktu widzenia projektowania eksperymentu jest określenie celu eksperymentu, którym może być

a. optymalizacja, tzn. znalezienie takich wartości czynników, dla których odpowiedź systemu (modelu) osiąga ekstremum

b. rozpoznanie charakteru zależności odpowiedzi systemu od czynników.

Jedną z metod znalezienia ekstremum odpowiedzi systemu w przypadku zmiennych dyskretnych, decyzyjnych jest zbadanie wszystkich możliwych kombinacji zmiennych, tzn. pełny eksperyment. W odniesieniu do zmiennych o charakterze ciągłym

(ilościowym) problem jest znacznie prostszy i możliwe jest stosowanie m.in. metod programowania matematycznego [5].

W tym wstępnym, pobieżnym przeglądzie problematyki projektowania eksperymentów należy jeszcze wspomnieć o zagadnieniu minimalizacji modelu, tzn. redukcji liczby czynników i doborze długości próbki. Do problemów tych powrócimy w następnych paragrafach.

3. ZBIEŻNOŚĆ STOCHASTYCZNA

Większość badanych procesów w rozważanej przez nas klasie problemów ma charakter stochastyczny. W wyniku eksperymentów symulacyjnych otrzymujemy żądane oceny statystyczne tych procesów w postaci wartości średnich i wariancji.

Wszystkie estymaty poszukiwanych wartości średnich obliczane są na podstawie próbek o skończonej liczności, uzyskiwanych z kilku przebiegów symulacyjnych. Zwiększenie dokładności estymacji, tzn. zwiększenie prawdopodobieństwa, że wyznaczone średnie będą bliższe rzeczywistym średnim, jest możliwe przez zwiększenie długości próbki. Miarą przypadkowych fluktuacji wielkości losowej jest jej odchylenie standardowe. Pamiętając, że estymata wartości średniej populacji jest samą zmienną losową, można przyjąć jako miarę dokładności ocen statystycznych wyznaczonych w czasie procesu symulacji, odchylenie standardowe estymaty wartości średniej $\delta_{\bar{x}}$. W przypadku n statystycznie niezależnych obserwacji

$$\delta_{\bar{x}} = \frac{\delta}{\sqrt{n}}$$

gdzie δ jest odchyleniem standardowym pojedynczej obserwacji (próbki)

Zatem aby dwukrotnie zmniejszyć przypadkowy błąd jakim obarczone są wyznaczone wartości średnie (np. średni czas oczekiwania w kolejce, średni czas odpowiedzi itp.) należy czterokrotnie zwiększyć długość próbki n (tak więc tę samą kolejkę należy obserwować czterokrotnie dłużej).

W efekcie utrzymywanie błędów na rozsądnie niskim poziomie może być związane z dużymi kosztami.

Taki stan rzeczy skłania do poszukiwania innych metod redukcji błędów. Na przykład można posługiwać się metodami Monte Carlo w celu zwiększenia efektywności eksperymentów symulacyjnych. Zasadą leżącą u podstaw techniki Monte Carlo jest pełne wykorzystanie informacji dotyczących struktury modelu i własności rozkładów prawdopodobieństwa zmiennych wejściowych celem zwiększenia dokładności pomiarów zmiennych wyjściowych.

Przedstawiony wyżej problem redukcji błędów znacznie się komplikuje w przypadku gdy mamy do czynienia z obserwacjami skorelowanymi. Dane wyjściowe uzyskiwane z symulacji są na ogół autoskorelowane (funkcja autokorelacji próbki $R_{s-t} \neq 0$ dla $s \neq t$). W tym wypadku nie można stosować wielu wygodnych metod statystycznych, prawdziwych dla niezależnych statystycznie obserwacji. Ignorowanie autokorelacji jest niedopuszczalne, ponieważ prowadzi do błędnych ocen. Odchylenie standardowe wartości średniej jest teraz określone wzorem

$$\delta_{\bar{x}} = \frac{1}{n} \sqrt{\sum_{s,t=1}^{n} R_{s-t}} = \frac{1}{\sqrt{n}} \sqrt{\sum_{s=1-n}^{n-1} (1 - |s|/n)\, R_s}$$

R_s - funkcja autokorelacji nieznormalizowana

Jak widać, aby uzyskać tę samą dokładność potrzebne są w tym przypadku znacznie dłuższe próbki. Obliczenia związane z wyznaczeniem funkcji autokorelacji są dla długich próbek bardzo czasochłonne. Aby określić minimalne n, przy którym $\delta_{\bar{x}} < \alpha$ (α - zakładany poziom błędu), nie można uniknąć obliczenia funkcji autokorelacji. W związku z tym badania i poszukiwania koncentrują się na takich algorytmach, które minimalizują koszty związane z tymi obliczeniami. Przykładem takiej metody może być metoda pośrednia określenia funkcji korelacji, drogą znajdowania modelu autoregresyjnego badanego przebiegu przypadkowego (sekwencji obserwacji). Model taki ma postać:

$$\sum_{s=0}^{r} b_s (X_{t-s} - \mu) = Y_t$$

Y_t - sekwencja statystycznie niezależnych zmiennych przypadkowych o jednakowym rozkładzie

$X_1 \ldots X_n$ - sekwencja obserwacji skorelowanych

$\mu = \frac{1}{n} \sum_{i=1}^{n} X_i$ - wartość średnia sekwencji obserwacji

b_s - współczynniki modelu autoregresyjnego

r - rząd modelu autoregresyjnego $1 \leqslant r \leqslant 5$

Funkcję autokorelacji badanej sekwencji obserwacji można wyrazić za pomocą współczynników b_s modelu autoregresyjnego.

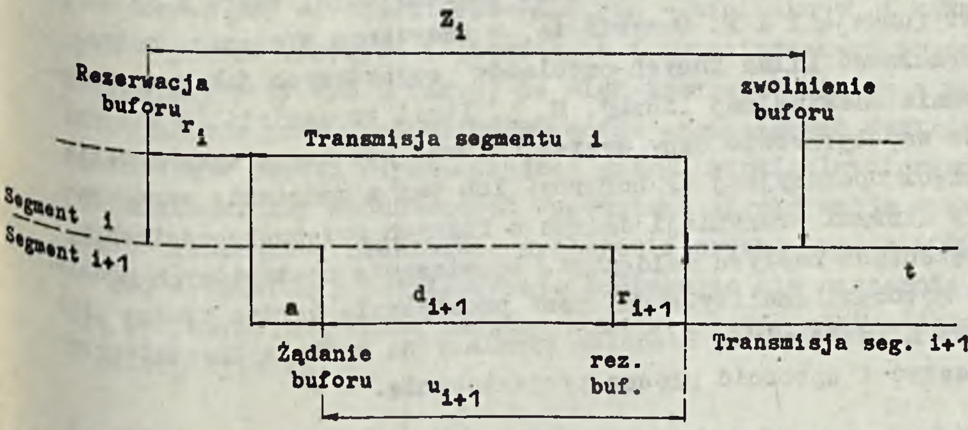
Powyższe zagadnienia omówione są szerzej w [2], [5], [6], [7], [8], [10].

4. PRZYKŁADY

Rozpatrzmy dwa przykładowe podsystemy systemu komutacji meldunków([3] oraz [11], [12]).

Przykład 1

Rys. 1 ilustruje zasadę działania systemu dynamicznej rezerwacji buforów pamięci operacyjnej dla segmentów meldunków wejściowych i wyjściowych. Zakładamy, że system ten współpracuje z M-liniami transmisji danych. Przyjmujemy ponadto, że znane są rozkłady prawdopodobieństwa długości meldunków oraz odstępów czasowych między kolejnymi meldunkami. Rozpatrywana tutaj technika segmentacji meldunków i przydziału buforów ma na celu zwiększenie wykorzystania pamięci operacyjnej. Jak wynika z rys. 1 należy dążyć do minimalizacji czasu blokowania $\bar{T}$.



Rys. 1.

z_i – czas zajętości buforu (dla segmentu i)
$\bar{z}$ – średni czas zajętości buforu
r_i – czas blokowania buforu (od momentu rezerwacji do chwili rozpoczęcia transmisji) dla segmentu i
$\bar{r}$ – średni czas blokowania buforu
d_i, $\bar{d}$ – czas (średni czas) oczekiwania na rezerwację buforu
a – opóźnienie żądania buforu dla segmentu (i+1) względem początku transmisji segmentu (i)

Działanie systemu zostaje zakłócone, gdy bufor dla segmentu i+1 nie zostanie zarezerwowany do chwili zakończenia transmisji segmentu i (strata informacji w przypadku meldunków odbieranych lub przerwa w transmisji w przypadku meldunków wysyłanych), tzn. gdy

$$d_i > u_i$$

Ze względów ekonomicznych rzadko dąży się do całkowitego wyeliminowania wspomnianych zakłóceń, a raczej do utrzymania prawdopodobieństwa ich występowania na zadanym, odpowiednio niskim poziomie. Zatem problem można sformułować jako poszukiwanie minimalnej liczby buforów, P_{MIN}, przy spełnieniu warunku

$$P_r \left\{ d_i > d_{KR} \right\} < \alpha$$

gdzie d_{KR} oznacza maksymalną wartość d, która nie powoduje zakłóceń, a α – zadany poziom prawdopodobieństwa.

Rozwiązanie tego zagadnienia jest trudne ze względu na to, że średni czas obsługi $\bar{z}$ jest zależny od $\bar{d}$, które z kolei jest funkcją $\bar{z}$ i P. Oczywiście, w omawianym systemie, można sformułować kilka innych problemów projektowych jak np. poszukiwanie maksymalnej liczby, M_{max}, linii transmisji, z którymi może współpracować dany system, biorąc pod uwagę ograniczenia pamięci operacyjnej (P buforów) lub też zagadnienie współpracy z liniami transmisji danych o różnych przepustowościach i natężeniach napływu meldunków. Rozwiązanie postawionych problemów metodami analitycznymi jest praktycznie bardzo trudne lub wręcz nieosiągalne. Technika symulacyjna pozwala znacznie przyśpieszyć i uprościć proces projektowania.

Na marginesie rozważań dotyczących omawianego wyżej przykładu warto wspomnieć o pewnym problemie praktycznym związanym z techniką symulacji: modelowaniu rzadkich zjawisk.

Aby móc poprawnie rozwiązać przedstawiony wyżej problem należy dokonywać pomiaru prawdopodobieństw występowania bardzo rzadkich zakłóceń w odbiorze lub nadawaniu informacji, wynikających z braku pamięci. Przyjmujemy, że wspomniane prawdopodobieństwo

$$P_r \left\{ d_i > d_{KR} \right\} \leqslant 10^{-3}$$

Zatem rozpatrywane zakłócenie występuje średnio rzadziej niż raz na tysiąc segmentów. Aby uzyskać wystarczająco wiarygodną estymatę rozważanego prawdopodobieństwa należałoby kontynuować eksperyment symulacyjny tak długo, aż rozpatrywane zdarzenie wystąpi wystarczającą ilość razy np. sto razy, co odpowiada symulacji obsługi 100 000 segmentów meldunków.

Jeżeli weźmiemy pod uwagę fakt, że do uzyskania zadowalających estymat pozostałych wielkości stochastycznych występujących w modelu, np. czasów $\bar{d}$, $\bar{z}$, wystarcza na ogół obserwacja obsługi $\sim$5000 segmentów, to widzimy, że należy poszukiwać przybliżonych pośrednich sposobów obliczenia szukanej wielkości prawdopodobieństwa. Wyznaczanie bezpośrednie jest niemożliwe ze względów ekonomiczno-czasowych.

Praktyczne rozwiązanie omówionego zagadnienia opiera się na pojęciu tzw. statystyk wartości ekstremalnych (tzn. własności statystycznych wartości ekstremalnych - minimalnych i maksymalnych - próbek losowych pochodzących z rozpatrywanego procesu). Korzystamy przy tym z faktu, że większość spotykanych w praktyce rozkładów prawdopodobieństwa można aproksymować funkcją wykładniczą w części odpowiadającej małym, marginalnym prawdopodobieństwom. Aby skonstruować wspomnianą aproksymację wystarcza często stosunkowo niewielkie wydłużenie próbki (np. z 5000 do 10 000 w omawianym przykładzie). Posługując się następnie funkcją aproksymującą można uzyskać poprawne oceny statystyczne rzadkich zjawisk.

Przykład 2. Wybór algorytmu obsługi odwołań do pamięci dyskowej

Złe wykorzystanie pamięci zewnętrznych (bębny, dyski) może w znacznym stopniu pogarszać efektywność systemów cyfrowych. Rozpatrzmy zatem zagadnienie wyboru optymalnego algorytmu obsługi odwołań do pamięci dyskowej (przedmiotem naszego zainteresowania są wyłącznie dyski z ruchomymi głowicami). Wybór dokonywany będzie spośród znanych algorytmów obsługi. Każdy z nich posiada swoiste, korzystne cechy w wąskim przedziale natężenia zgłoszeń. Problem polegać więc może na dokonaniu kombinacji dwóch, lub większej liczby algorytmów podstawowych tak, aby zapewnić żądane cechy w szerokim zakresie zmienności natężeń odwołań.

Przyjmijmy przykładowo następującą funkcję - kryterium:

$$Q_i = T^a_{S_i} \; (bW_i + c\,\delta_{W_i})$$

$T_{S_i}(\lambda)$ - średni czas obsługi odwołania do dysku (dla i-tego algorytmu)

$W_i(\lambda)$ - średni czas oczekiwania odwołania do dysku od momentu przybycia do kolejki do chwili zakończenia obsługi (dla i-tego algorytmu)

$\delta^2_{W_i}$ - wariancja czasu oczekiwania (dla i-tego algorytmu)

λ - średnia częstotliwość napływu odwołań do dysku w ciągu 1 s.

a, b, c - współczynniki wagi

Mając kryterium przystępujemy do eksperymentu symulacyjnego z poszczególnymi algorytmami. Z każdego przebiegu symulacyjnego otrzymujemy wartości Q_i. Następnie szukamy ekstremum (minimum) Q_i. Indeks i, przy którym funkcja-kryterium osiąga ekstremum, określa algorytm optymalny ze względu na to kryterium.

5. PEWNE ASPEKTY OPTYMALIZACJI PROJEKTOWANYCH SYSTEMÓW

Problemy przedstawione w powyższych przykładach można próbować rozwiązywać metodami tradycyjnymi w sposób iteracyjny: wykonuje się eksperyment symulacyjny, analizuje uzyskane wyniki, na podstawie których można określić nowe wartości parametrów modelu, dla kolejnego eksperymentu. Metoda ta stosunkowo dobrze zdaje egzamin w przypadkach strukturalnie prostych modeli o niezbyt dużej liczbie zmiennych. Gdy mamy do czynienia z problemem zbiorczym wielowymiarowym trudno jest, nie znając wyraźnie zależności wiążących rozpatrywane zmienne z kryterium oceny, określać w sposób świadomy, efektywny kierunki poszukiwań lepszych rozwiązań. Tak więc procedurę poszukiwania dopuszczalnego lub najlepszego rozwiązania należy wzbogacić o możliwości i środki identyfikacji, na bieżąco, na podstawie zgromadzonych informacji, podstawowych związków i zależności między zmiennymi modelu. W tej sytuacji problem może się okazać obliczeniowo bardzo czasochłonny, komplikując i ograniczając tym samym prace projektowe.

Spróbujmy zatem sformułować podstawowe wymagania dla systemu programowego, który w sposób automatyczny lub być może półautomatyczny przy współpracy analityka umożliwiłby uzyskanie rozwiązania zbliżonego do optymalnego. Formułując model symulacyjny, który będzie wykorzystany do celów predykcji charakterystyk użytkowych, zwłaszcza w przypadku gdy nie dysponujemy odpowiednim doświadczeniem w projektowaniu rozpatrywanej klasy systemów, łatwo popada się w nadmierną drobiazgowość opisu wierząc, że w ten sposób łatwiej uzyskuje się zadowalające wyniki eksperymentu. Być może postępowanie to jest wynikiem nawyków charakterystycznych dla programistów: aby dobrze zapro-

gramować problem obliczeniowy należy wnikliwie rozpatrzyć wszystkie szczegóły, możliwości, kombinacje itp. Postępowanie to, przeniesione na grunt modelowania, daje na ogół wyniki negatywne, powoduje trudności obliczeniowe. Model powinien zawsze możliwie syntetycznie uwzględniać aspekty badanego obiektu (nie wszystkie), interesujące jedynie z punktu widzenia rozwiązywanego problemu. Należy tutaj raz jeszcze przypomnieć, że model jest zawsze narzędziem podporządkowanym celom eksperymentalnym.

Mając na uwadze automatyzację eksperymentalnej fazy projektowania należy nadmierną drobiazgowość modelu (duża liczba zmiennych nie wpływających na kryterium oceny) uznać za szkodliwą. Z drugiej strony trzeba pamiętać, że analityk konstruujący model systemu może nie wiedzieć lub nie spodziewać się istnienia pewnych oddziaływań i związków wewnętrznych (oddziaływania te mogą mieć charakter zanikający, przejściowy, wystąpienie efektów może być znacznie opóźnione w stosunku do przyczyn). Tak więc w eksperymencie symulacyjnym uwzględnić należy etap przygotowawczy, mający na celu testowanie i minimalizację modelu. Omówione wyżej badania dokonywane są za pomocą metod analizy statystycznej: analizy czynnikowej oraz analizy wariancji.

Dysponując odpowiednio przygotowanymi modelami można przystąpić do następnej fazy eksperymentu: wyboru odpowiednich metod poszukiwania rozwiązania optymalnego. Podstawowym czynnikiem rzutującym na tok dalszego postępowania jest charakter zmiennych występujących w modelu.

Analizując przykład 1 stwierdzamy, że występują w nim wyłącznie zmienne o charakterze ciągłym. W tej sytuacji można planować wykorzystanie metod programowania matematycznego (np. programowania liniowego lub metod najszybszego spadku) do wyznaczania poszukiwanego rozwiązania optymalnego. Nasuwa się więc następny problem racjonalnego wyboru odpowiedniej metody lub klasy metod najbardziej przydatnych i odpowiednich do rozwiązania naszego zagadnienia.

Zrozumiałe jest, że ze względów czasowo-ekonomicznych wybierać będziemy metody szybko zbieżne, tzn. gradientowe [5]. Zasa-

dę postępowania w przypadku stosowania wspomnianych metod gradientowych streścić można w następujących punktach:

1. wybieramy punkt startowy (projekt początkowy) dla procedury

2. dokonujemy liniowej aproksymacji (hiperpłaszczyzna aproksymująca) funkcji celu (odpowiedź systemu) w otoczeniu punktu startowego. Wykorzystuje się w tym celu standardowe techniki statystyczne (metoda aproksymacji średniokwadratowej). W wyniku takiego postępowania wyznaczamy kierunek poszukiwań rozwiązania optymalnego.

3. wzdłuż wyznaczonego uprzednio kierunku dokonujemy poszukiwań dążąc do ustalenia punktu ekstremalnego (minimum lub maksimum), tzn. takiego rozwiązania, że istnieje małe prawdopodobieństwo polepszenia uzyskanych rezultatów. Należy w tym miejscu zwrócić uwagę na poważny problem praktyczny doboru długości kroku dla poszukiwań kierunkowych. Zbyt mały krok powoduje znaczny wzrost obliczeń, wolniejsze posuwanie się w kierunku optimum; zbyt duży krok może doprowadzić do ominięcia lub trudności w zlokalizowaniu ekstremum. Nie istnieją ogólne reguły doboru długości kroku, które można by zaakceptować do szerokiej klasy problemów. Praktyczny dobór odbywa się zwykle w sposób doświadczalny.

4. otrzymane w p. 3 ekstremum kierunkowe staje się punktem startowym do nowego cyklu iteracyjnego. Proces poszukiwania rozwiązania optymalnego przerywa się, gdy zostaną spełnione warunki wskazujące, że prawdopodobieństwo otrzymania lepszego rozwiązania jest odpowiednio małe. W przypadku gdy aproksymacja liniowa nie wystarcza do prawidłowego wyznaczenia dalszych kierunków poszukiwań ekstremum, np. w przypadkach punktu siodłowego lub w pobliżu punktu ekstremalnego, wykorzystuje się aproksymację kwadratową.

Wspomniane metody gradientowe wymagają różniczkowalności funkcji celu (kryterium). Może się więc zdarzyć (choć wydaje się, że niezbyt często), że będziemy zmuszeni korzystać z metod bezgradientowych. Jedną z nich najbardziej naturalną, którą należy wspomnieć w tym miejscu jest metoda uzmiennienia pojedynczych czynników. Dokonując zmian wartości pojedynczej zmiennej przy pozostałych ustalonych dąży się, oczywiście, do osiągnięcia ekstremum (kierunek poszukiwania ekstremum jest więc w tym przypadku zgodny z kierunkiem wybranej osi współrzędnych). Następnie ustalając poprzednią zmiennej na poziomie osiągniętej dotychczas wartości ekstremalnej (największej lub najmniejszej), uzmiennia się inny czynnik. Po wykonaniu opisanych prób dla wszystkich n-czynników powraca się do czynnika pierwszego i ponawia się cykl poszukiwań. Proces optymalizacji zostaje wstrzymany, gdy zmiany żadnego czynnika nie powodują polepszenia wartości funkcji celu. Jak widać jest to metoda bardzo prosta, wygodna do stosowania. Jej podstawową wadą jest ograniczenie kierunków poszukiwań (kierunki osi głównych układu współrzędnych). Może to prowadzić do błędnych wyników, tzn. zatrzymania się procedury daleko od punktu ekstremalnego, np. w przypadku "ostrego grzbietu" funkcji celu, nie równoległego do żadnej osi współrzędnych. Omówiona metoda, jak łatwo zauważyć, nie odznacza się szybką zbieżnością. Liczba eksperymentów jaką należy wykonać, aby zbliżyć się do optimum jest bardzo duża. Problematyka wykorzystania programowania matematycznego do optymalizacji w dziedzinie procesów dyskretnych, z wykorzystaniem eksperymentów symulacyjnych, jest aktualnie w skali światowej bardzo mało rozpoznana. Nieliczne ośrodki amerykańskie przeprowadzają prace badawcze w tej dziedzinie i na podstawie fragmentarycznych doniesień można wnioskować, że wyniki są pozytywne. Zatem, trudno obecnie oceniać przydatność poszczególnych klas algorytmów. Wydaje się, że podstawowym problemem limitującym szerszą możliwość zastosowań omawianych metod są koszty. Prowadzenie eksperymentów optymalizacyjnych dla złożonych systemów jest bardzo czasochłonne, przy czym czas ten zużywany jest głównie na przebiegi symulacyjne. Dotychczasowe doświadczenia wskazują, że traktując indywidualnie każdy z rozwiązywanych problemów (mowa o problemach średniej wielkości), można dla

wielu z nich, na ogół drogą odpowiednich dekompozycji, aproksymacji, przekształceń uzyskać zadowalające rezultaty.

Pozostał jeszcze problem optymalizacji z ograniczeniami. Wszystkie powyższe uwagi zachowują swą słuszność. W przedstawionym ujęciu problemu wykorzystanie metod optymalizacji z ograniczeniami (np. metody wykorzystujące funkcję kary) prowadzi jedynie do dalszego zwiększenia liczby wymaganych obliczeń (przebiegów symulacyjnych).

Jeżeli w rozpatrywanym problemie występują zmienne typu dyskretnego (decyzyjne), jak np. w przykładzie 2, nie można planować wykorzystania metod opisanych wyżej. Wartości liczbowe takich zmiennych nie mają żadnego związku z wartością przyjętego kryterium oceny (np. indeksy algorytmów, które są reprezentowane w modelu przez pewną zmienną). W tej sytuacji, aby dokonać wyboru najlepszego rozwiązania należy korzystać z metod analizy czynnikowej [5].

6. ZAKOŃCZENIE

Artykuł porusza niektóre istotne problemy, które wyłaniają się w czasie projektowania złożonych systemów za pomocą modelowania symulacyjnego.

Oczywiście jest to tylko część problemów. Nie omówiono tu takich istotnych spraw jak dobór języka symulacyjnego do modelowania wybranego obiektu, modelowanie środowiska, w którym symulowany system jest "zanurzony".

Artykuł koncentruje się głównie na etapie eksperymentu z gotowym modelem. Autorzy zwracają uwagę na możliwość zautomatyzowania eksperymentu, tzn. powiązania przebiegów symulacyjnych z analizą wyników i optymalizacją parametrów systemu.

Literatura

[1] CONWAY R.W.: Some Tactical Problems in Digital Simulation, Management Science X, 1963, nr 1, s. 47-61.

[2] CONWAY R.W., JOHNSON B.M.: Some Problems of Digital Machine Simulation, Management Science VI, 1959, nr 1.

[3] BRICAULT J.P., DELGALVIS I.: On Teleprocessing System Design, IBM Syst. Journal, 1966, vol. 5, nr 3, s. 66.

[4] BURDICK D.S., NAYLOR T.H.: Design of Computer Simulation Experiments for Industrial Systems, Comm ACM IX, 1966, nr 5, s. 329-338.

[5] EMSHAFF J.R., SISSON R.: Design and Use of Computer Simulation Models, The Macmillan Company, 1970,

[6] FISHMAN G.S.: Digital Simulation: Input-Output Analysis, Tech. Memo RM 5540PR Santa Monica Calif. Rand Corp., 1968.

[7] FISHMAN G.S.: Problems in the Statistical Analysis of Simulation Experiments, The Comparison of Means and the Length of Sample Records, Comm ACM X, 1967, nr 2, s. 94-99.

[8] FISHMAN G.S., KIVIAT Ph.: The Analysis of Simulation-Generated Time Series, Management Science XIII, 1967, nr 7, s. 525-557.

[9] GRAHAM R.M.: Advanced Course on Software Engineering Ed by FL Bauer Lecture Notes in Economics and Mathematical Systems 81 Springer - Verlag, 1973.

[10] GORDON G.: System Simulation, Prentice-Hall INC, 1969.

[11] TEOREY T.J., PINKERTON T.B.: A Comparative Analysis of Disc Scheduling Policies, CACM, 1972, vol. 15, nr 3.

[12] WEINGARTEN A.: Storage Requirement for a Message Switching Computer, IEEE Trans. of The Professional Technical Group on Communication Systems, 1964, vol. CS-12, nr 2,

МОДЕЛИРОВАНИЕ И СИМУЛЯЦИОННЫЙ ЭКСПЕРИМЕНТ В ПРОЕКТИРОВАНИИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Резюме

Сложность современных вычислительных систем, обусловленная развитием функциональных возможностей вычислительных средств, высокие требования относительно надежности матобеспечения заставляют проектировщиков широко использовать и развивать инструменты программирования, позволяющие быстро проверять и оценивать в количественном отношении проектные концепции.

В настоящей разработке затрагиваются существенные проблемы, появляющиеся в процессе проектирования сложных систем путем симуляционного моделирования. Эти проблемы проиллюстрированы практическими примерами.

MODELLING AND SIMULATION EXPERIMENT IN COMPUTER SYSTEMS DESIGN

Summary

The increase of current digital systems complexity resulted from the development of functional possibilities of hardware and user's requirements concerning software effectiveness encourage software designers to use and develop programming tools for fast verification and evaluation of design concepts.

This paper presents basic problems arising during complex systems designing which employes simulation modelling. Problems are discussed on the base of some experiments.

ALGORYTMY, Zeszyt Specjalny, 1974, 121-130

681.322.06:
061.3/438/"1973"

ANALIZA SYSTEMÓW LICZĄCYCH METODĄ SYMULACJI

Józef MAROŃSKI

Centrum Obliczeniowe PAN

Pracę złożono 10.I.1974

W referacie zostaną przedstawione pewne problemy zastosowania metody symulacji do badania systemów liczących. Między innymi będą omówione takie zagadnienia jak: weryfikacja modeli symulacyjnych, wiarygodność wyników symulacji, adaptacja systemu liczącego do potrzeb ośrodka obliczeniowego itp. W skrócie omówione zostaną również najważniejsze prace dotyczące badań systemów liczących z wykorzystaniem metody symulacji.

S p i s t r e ś c i

1. WSTĘP

W momencie kiedy maszyny cyfrowe stały się powszechnym narzędziem w badaniach naukowych oraz zastosowaniach komercjalnych, ogromnego znaczenia nabrał problem szacowania i wyboru optymalnych dla danych zastosowań systemów liczących. Problem ten jest szczególnie ważny podczas instalowania systemów do przetwarzania danych. Systemy te z reguły wieloprogramowe,

często wieloprocesorowe stają się tak złożone, że nawet najbardziej doświadczony projektant nie jest w stanie dokonać ich oszacowania w sposób intuicyjny. Użytkownicy coraz częściej poszukują narzędzi do porównywania konkurencyjnych propozycji różnych firm komputerowych.

Referat stanowi pewnego rodzaju przegląd zagadnień, w rozważaniu których wykorzystuje się lub można wykorzystać jedną z najodpowiedniejszych dla maszyn cyfrowych metod, metodę symulacji.

Poza tym omówione będą krótko dwa z podstawowych zagadnień związanych z metodą symulacji: zagadnienie weryfikacji modeli symulacyjnych oraz zagadnienie wiarygodności wyników symulacji.

Na zakończenie omówione zostaną różne rozwiązania zarówno sprzętowe jak i programowe opracowane przez różne firmy celem przeprowadzania analiz działania różnych systemów.

2. ROZWIĄZYWANIE PROBLEMÓW OPROGRAMOWANIA METODĄ SYMULACJI

Z zagadnieniem analizy i oceny systemów cyfrowych w bardzo ścisły sposób związane jest ustalenie odpowiednich kryteriów tej oceny. Od dawna kryteria stosowane do tych celów ulegają ciągłym zmianom, a najczęściej spotykana jest tzw. moc systemu oraz koszt jego instalacji. Ponieważ istnieje ścisła zależność pomiędzy mocą systemu a wspomnianymi kosztami instalacji i eksploatacji nasuwa się całkiem naturalne dążenie do pewnego rodzaju optymalizacji tych wielkości. Oczywiście optymalizacja taka jest możliwa wówczas, kiedy jesteśmy w stanie określić odpowiednie miary analizowanych wielkości.

Optymalizacja systemu jest możliwa tylko wtedy, kiedy istnieje możliwość zmiany wewnętrznej struktury danego systemu; kiedy np. można rezygnować z pewnych funkcji systemu kosztem zwiększenia mocy tego systemu itp.

Metoda symulacji ma szczególne znaczenie podczas instalowania systemów przetwarzania danych, czy systemów służących do

sterowania obiektami przemysłowymi. W systemach przetwarzania danych, szczególnie tych złożonych z programów parametryzowanych, symulacja służy do ustalania najbardziej odpowiednich parametrów dla danego środowiska. Dobierane są parametry określające zbiory danych oraz rodzaje operacji wykonywanych na tych zbiorach w tym środowisku. Podobnie przedstawia się problem szacowania systemów do sterowania procesami, w których można dokonywać zmian harmonogramów działania programów obsługi itp.

W ostatnim czasie pojawiło się wiele problemów natury znacznie ogólniejszej niż wspomniano wyżej. Jednym z nich jest tzw. "kryzys software'owy", na który składają się dwa zasadnicze problemy nierozwiązane dotychczas w sposób definitywny. Problem pierwszy to niezadowolenie użytkowników, którzy uważają, że obecnie dostępne systemy liczące nie zaspokajają ich potrzeb. Pretensje te, występujące niezależnie od etapu rozwoju systemów liczących, można dość łatwo wytłumaczyć. Nawet częste kontakty projektantów systemów z użytkownikami, które niektórzy postulują, w sposób generalny nie poprawiają sytuacji pod tym względem. Bowiem potrzeby różnych użytkowników, chociaż dotyczą tego samego profilu obliczeń, mogą być zasadniczo różne. Poza tym system liczący zadowalający jednego użytkownika, z którym projekt konsultowano może być nieodpowiedni dla wszystkich pozostałych użytkowników. Jednym z odpowiednich rozwiązań tego problemu wydaje się budowa systemów otwartych, w skład których wchodziłyby takie elementy oprogramowania i sprzętu, które zaspokajają bieżące potrzeby danego użytkownika. Dobór tych elementów, szczególnie przy bardziej złożonych systemach, nie jest możliwy bez zastosowania odpowiednich metod. Jedną z takich metod jest metoda symulacji. Nie należy z tego wnioskować, że metoda ta zlikwiduje wspomniany problem "kryzysu software'owego", tym niemniej może ona pomóc w jego złagodzeniu. Oczywiście w tym celu należy przyjąć odpowiednie metody projektowania systemów liczących i operacyjnych. Szczególnie niezbędna okazuje się tu struktura modularna, pozwala ona bowiem na dołączanie, ewentualnie wymianę odpowiednich dla danego użytkownika modułów programowych.

Wydaje się, że metoda symulacji może również dopomóc w złagodzeniu drugiego problemu "kryzysu software'owego". Mianowicie może przyczynić się do obniżenia kosztów wytwarzania oprogramowania, jeśli będzie ono mogło się zmieniać w zależności od potrzeb użytkowników, natomiast jego elementy składowe będą pobierane ze stałego zbioru programów, a nie projektowane dla każdego typu użytkowników niezależnie. Szczególnie jest to ważne podczas oprogramowywania serii maszyn powiązanych wspólnym oprogramowaniem. Symulacja pozwala na dobór odpowiednich zbiorów programów stanowiących np. systemy operacyjne.

Jeszcze innym problemem, może mniej związanym z "kryzysem software'owym" lecz równie ważnym, jest problem samodzielnego wprowadzania zmian do systemu przez różnych użytkowników. Próby takie w ostatnim czasie notujemy coraz częściej. Aby ułatwić dokonywanie takich zmian konieczne są pewne środki pozwalające na obserwację współdziałania poszczególnych elementów oprogramowania. I tak można wbudować do systemów odpowiednie programy "podglądające" oraz pozwalające na odpowiednie zmiany w oprogramowaniu. Wydaje się, że dopiero połączenie tych ułatwień z metodą symulacji może dać odpowiednie efekty. Można bowiem dzięki temu obserwować różne aspekty współdziałania systemu przed i po ewentualnych zmianach dokonywanych przez użytkownika w systemie i jego modelu symulacyjnym. W przypadku kiedy producent dostarcza razem z systemem jego model symulacyjny, koszty związane ze zmianą takiego systemu znacznie się obniża.

Z tym samym zagadnieniem wiąże się sprawa testowania wewnętrznego systemów. Jeśli wspomniany model systemu jest dostatecznie szczegółowy, wówczas testowanie można znacznie przyspieszyć przez testowanie modelu, traktując testowane elementy bardziej szczegółowo, a elementy już przetestowane lub w danym momencie nieistotne bardziej ogólnie. Taki model symulacyjny może stanowić równocześnie pewnego rodzaju opis dokumentacyjny danego systemu.

3. WERYFIKACJA MODELI SYMULACYJNYCH

Zastosowanie metody symulacji jak i innych metod nastręcza pewne trudności, inaczej mówiąc, stwarza pewne problemy. Jednym z nich jest weryfikacja modeli, czyli stwierdzenie zgodności modeli z systemami rzeczywistymi.

W zastosowaniach tej metody można zauważyć dwa zasadnicze sposoby weryfikacji modeli. Pierwszy z nich polega na stwierdzeniu zgodności modelu systemu z samym systemem. Z reguły odbywa się to przez porównanie pewnych wyników otrzymanych w czasie działania systemu rzeczywistego z wynikami otrzymanymi w czasie działania modelu tego systemu. Jeśli wyniki będą zgodne z odpowiednią tolerancją, wówczas możemy stwierdzić, że model właściwie odzwierciedla system. Oczywiście taka weryfikacja jest możliwa tylko wówczas, kiedy istnieje system rzeczywisty i możemy uzyskać pewne obserwacje podczas jego działania.

Inaczej przedstawia się sytuacja w przypadku, kiedy system rzeczywisty nie istnieje, jest w stadium projektu, jak również w przypadku kiedy musimy, np. ze względu na wielkość symulowanego systemu, ograniczyć się do uwzględnienia w modelu tylko niektórych elementów systemu w sposób szczegółowy a wyeliminowania częściowo lub całkowicie innych. W tych przypadkach model weryfikowany jest najczęściej tzw. metodą prób i błędów. Wykonuje się działania symulacyjne z uwzględnieniem danego elementu lub z jego szczegółowym modelem i bez tego elementu lub z jego modelem bardziej ogólnym. Porównanie uzyskanych wyników pozwala zadecydować czy dany element ma istotne znaczenie i jak należy go traktować w dalszych badaniach. Powtarzając ten proces wielokrotnie dla coraz to innych elementów modelu możemy w efekcie uzyskać model całego systemu, który będzie najodpowiedniejszy do dalszych badań.

4. WIARYGODNOŚĆ WYNIKÓW SYMULACJI

Innym, niemniej ważnym, problemem występującym w symulacji jest wiarygodność uzyskanych wyników.

Podczas każdego działania symulacyjnego generowane są pewne wielkości zwane wartościami cech modelu danego systemu*). Ponieważ z reguły symulacja opiera się na pojawieniu się zdarzeń w sposób losowy, w celu oszacowania jakiejś wielkości dokonuje się odpowiedniej liczby działań symulacyjnych. Wobec tego dla jednej cechy otrzymujemy wiele wartości z poszczególnych działań. W takim przypadku posługujemy się oszacowaniami wartości średnich poszczególnych cech. Ponieważ wartości średnie tych cech są również wielkościami losowymi, stosujemy tzw. szacowanie przedziałem.

Opierając się na centralnym twierdzeniu granicznym, które mówi, że dla dostatecznie dużej liczby prób n, teoretyczny rozkład z próby można z dużą dokładnością aproksymować za pomocą rozkładu normalnego, możemy twierdzić z prawdopodobieństwem $1-\alpha$, że wielkość błędu $\bar{x}-\mu$, który popełniamy przyjmując $\bar{x}$ jako oszacowanie wartości średniej μ, będzie mniejsza niż $Z_\alpha \cdot \frac{\sigma}{\sqrt{n}}$, gdzie Z_α oznacza współczynnik ufności, a σ oznacza odchylenie standardowe populacji, które zastępujemy odchyleniem standardowym z próby

$$s = \sqrt{\frac{1}{n}\sum (x_i - \bar{x})^2} .$$

W większości przypadków z zastosowania metody symulacji wyciągamy wnioski na podstawie wartości średnich odpowiednich cech. W takiej sytuacji jednym z ważniejszych elementów procesu symulacji jest przeprowadzenie testu istotności różnic pomiędzy wartościami średnimi odpowiednich cech dla różnych przebiegów symulacyjnych. Chodzi w tym przypadku o stwierdzenie, czy różnica między średnią wartością cechy dla jednego działania symulacyjnego i taką samą średnią dla innego działania przy zmienionych warunkach jest wielkością przypadkową, czy rzeczywiście zależy od zmiany warunków. Stosuje się w takich przypadkach zwykle test t-Studenta oparty na wzorze

$$t^o = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{S_1^2 + S_2^2}{n-1}}}$$

*) Terminu tego używamy w tym samym sensie co przy badaniach statystycznych systemów rzeczywistych.

gdzie $\bar{x}_1$ i $\bar{x}_2$ oznaczają odpowiednio testowane średnie, a S_1^2 i S_2^2 odpowiednie wariancje.

Tak obliczoną wartość testową porównuje się z wartością graniczną rozkładu t dla 2n-2 stopni swobody przy założonym poziomie ufności. Jeśli moduł obliczonej wartości testowej jest większy od wartości granicznej, wówczas z zadanym prawdopodobieństwem możemy twierdzić, że poszczególne średnie różnią się między sobą w sposób istotny a różnica nie jest dziełem przypadku.

Omówiona metoda testowania istotności różnic pomiędzy odpowiednimi średnimi może służyć również podczas podejmowania decyzji, w trakcie budowy modelu, czy dany element ma istotne znaczenie, czy też można go odrzucić.

5. PRZEGLĄD SYSTEMÓW ANALIZY I SZACOWANIA SYSTEMÓW LICZĄCYCH

Na zakończenie dokonamy pewnego przeglądu istniejących systemów badania oprogramowania i sprzętu. Systemy te opierają się na różnych zasadach. Istnieją systemy sprzętowe działające na zasadzie urządzeń elektronicznych podłączanych bezpośrednio do systemów liczących, specjalne programy włączane do systemu zbierające i analizujące dane uzyskiwane w czasie pracy analizowanego systemu, jak również symulatory imitujące pracę badanych systemów.

Wydaje się, że najodpowiedniejsze w takich szacowaniach są systemy pierwszego i ostatniego typu. Urządzenia sprzętowe bowiem z reguły nie obciążają jednostki centralnej w badanym systemie, nie ma dodatkowych narzutów na zarządzanie takim analizatorem. Symulatory z kolei działają w sposób niezależny od badanego systemu. Wykorzystuje się w nich tylko informacje uzyskane z tych systemów w trybie "off-line".

Natomiast analizatory systemów zbudowane na zasadzie programów dają obraz nieco sfałszowany wskutek tego, że zwiększają się narzuty pracy jednostki centralnej na zarządzanie i wykonywanie tych programów, jak również zmniejsza się obszar wol-

nej pamięci operacyjnej. Z reguły takie analizatory zbudowane są w sposób dwuczęściowy. Pierwsza z nich pracuje w czasie działania analizowanego systemu i rejestruje różnego rodzaju dane, które następnie w sposób niezależny przetwarza druga część analizatora, sporządzając odpowiednie raporty.

Przedstawiony niżej przegląd nie obejmuje na pewno wszystkich zbudowanych systemów analizy i szacowania systemów liczących. Jednak daje pewien obraz tendencji jakie panują w tej dziedzinie.

Systemy oparte na urządzeniach sprzętowych oferują następujące firmy:

- Applied Computer Technology, Inc., system CPM II,
- Applied Systems Div., system X-RAY,
- Computer Programming a. Analysis Inc., system CPA 7700,
- Computer Programming and Consultation, system COPAC,
- COMRESS, system Dynaprobe/Dynapar,
- IBM, system IS/PAR,
- University of California at Los Angeles, system SNUPER.

Systemy oparte na monitorach programowych oferują:

- Boole a. Babbage, system CUE,
- Computer Management Sciences, system M-Test,
- Computer Synectics, Inc., system SUM,
- IBM, system SIPE,
- Lambda Corporation, system LEAP,
- Standard Linear Accelerator Center, system PROGLOOK.

Trzeci rodzaj analizatorów tzn. symulatory oferują następujące firmy:

- Applied Data Research, Inc., system SAM,
- Applied System Division, system CASE,
- COMRESS, system SCERT,
- IBM, system AMAP,
- Nippon Electric Company, system PACSS

oraz wiele innych symulatorów podawanych w literaturze bez nazwy. Do najbardziej znanych można zaliczyć symulatory Scherra, Goldsteina, Fine'a i McIsaaca oraz Katza.

Należy sądzić, że w najbliższym czasie nastąpi pewnego rodzaju intensyfikacja prac badawczych dotyczących wypracowywania kryteriów oraz metod badawczych zarówno systemów liczących i operacyjnych, jak również niezależnych programów.

АНАЛИЗ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ПУТЕМ СИМУЛЯЦИОННОГО МЕТОДА

Резюме

Развитие вычислительных систем все еще отстает от развития применения этих систем в разных областях. Чтобы ускорить развитие вычислительных систем, применяются самые современные методы исследований и проектирования, в том числе и симуляционный метод.

В настоящем труде представлены некоторые проблемы использования симуляционного метода для исследований вычислительных систем. В частности, описываются такие вопросы, как проверка симуляционных моделей, достоверность результатов симуляции, приспособление вычислительной системы к потребностям вычислительного центра и т.п. Кратко обсуждаются также главные труды, касающиеся исследования систем при использовании метода симуляции.

COMPUTING SYSTEMS ANALYSIS BY A SIMULATION METHOD

Summary

The development of computing systems is still late in comparison to the development of applications of these systems in various fields. In order to accelerate this development the most modern research and design methods were developed, among others simulation methods.

Some problems connected with the application of simulation methods to the analysis of computing systems are presented in the paper. In particular, such problems as simulation models verification, reliability of simulation results, and adjusting of a system to the real needs of a computing centre, etc., are discussed. A brief review of the most important results obtained in the application of simulation methods for analyzing the computing systems is given.

IV. PROBLEMY PROGRAMOWANIA I DOKUMENTACJI

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

ALGORYTMY, Zeszyt Specjalny, 1974, 133-155 681.322.06:
061.3/438/"1973"

MODULARNE PROGRAMOWANIE SYSTEMÓW OPERACYJNYCH

Hanna MYSIOR
Jerzy MYSIOR

Instytut Maszyn Matematycznych

Pracę złożono 10.I.1974

Omówiono problemy projektowania i realizacji elastycznych systemów operacyjnych. Przy projektowaniu określa się hierarchiczną warstwową strukturę systemu operacyjnego i następnie warstwy dzieli się na mniejsze elementy - moduły. Wymagany jest sformalizowany opis modułów, na podstawie którego można je zaprogramować i stosować, oraz opis sposobu ich łączenia w konkretnym systemie. Opis ten stanowi wystarczającą dokumentację systemu operacyjnego. Wewnętrzna struktura modułów nie jest określona. Mogą one być realizowane przy użyciu różnych języków programowania.

S p i s   t r e ś c i

1. WSTĘP

Współcześnie produkowane i rozpowszechniane systemy operacyjne osiągnęły taki stopień złożoności, że ich poznanie choćby w celu zdobycia umiejętności pełnego korzystania z dostarczanych przez nie udogodnień przekracza możliwości przeciętnego programisty. I tak jak kiedyś brak udogodnień w korzystaniu z maszyn narzucał przeciętnemu użytkownikowi konieczność korzystania z pośrednictwa zawodowego programisty, tak obecnie nadmiar ich prowadzi do tego samego. Sytuacja komplikuje się, gdy użytkownik chce korzystać w tym samym czasie z systemów liczących wyprodukowanych i oprogramowanych przez różnych producentów lub też chce po doświadczeniach z jednym systemem zacząć korzystać z innego. Jak trafnie zauważa Cox [4] jedyną wspólną cechą współczesnych dużych systemów operacyjnych jest to, że są one ogromne. Do oczywistych nonsensów należy zaliczyć np. fakt, że języki typu job control language, które w każdej maszynie spełniają ten sam cel, a mianowicie wiążą logiczne urządzenia wejścia-wyjścia programu z fizycznymi i mówią o tym jakie programy i w jakiej kolejności mają być wykonywane, zaprojektowane celem ułatwienia życia programiście, są przeważnie skomplikowane, nie osiągnęły standaryzacji często w obrębie tego samego systemu a już z reguły są różnie zaprojektowane w różnych systemach.

Przyczyną tego stanu rzeczy jest obraz użytkownika jaki wytworzyli sobie producenci. Użytkownik powinien z radością przyjmować to co mu się przekazuje, gdyż wśród dostarczonych ułatwień są na pewno i te, na które liczy.

Prześledźmy przyczyny i skutki takiego rozumowania: początkowo na skutek ogromnych możliwości sprzętowych potrzeby użytkowników były niewielkie. Skromny dorobek programowy przy pojawieniu się nowej maszyny mógł być odtworzony stosunkowo niewielkim nakładem kosztów. Ten dorobek programowy narastał wraz z rozszerzaniem się możliwości sprzętowych, jednocześnie rosły potrzeby, co z kolei wymagało nowego sprzętu. Pomiędzy użytkownikiem a maszyną stanął system operacyjny, którego zadaniem jest ułatwić użytkownikowi wykorzystanie sprzętu. Ukształtowała się praktyka, żeby uprzedzając życzenia użytkownika dać mu możli-

wie najwięcej środków programowych do urzeczywistnienia wszelkich możliwych życzeń. Oczywiście zawsze jednak znajdzie się taka potrzeba, której dostarczonymi środkami zaspokoić nie można. A jeśli nawet te odpowiadające potrzebom środki istnieją, to szanse odnalezienia w ogromnej masie tego właśnie, co jest potrzebne i możliwości skorzystania z czegoś, co stosuje się ogromnie rzadko, a więc nie ma się biegłości w używaniu, są problematyczne. Stosuje się różne środki zaradcze np. w postaci angażowania wysoko wykwalifikowanych programistów specjalnie po to, by umieli tłumaczyć język potrzeb użytkownika na język możliwości systemu. Jest to niezmiernie kosztowne, przede wszystkim jednak szkoda marnować zdolnych ludzi mogących wykonywać o wiele bardziej wartościowe prace [4].

Niektórzy użytkownicy próbują na własną rękę mniej lub bardziej udolnie modyfikować system, upodabniając go do tych, z którymi mieli do czynienia dotychczas.

Tak więc wyobrażanie sobie czego użytkownik może potrzebować jest rozumowaniem fatalnym w skutkach.

Takiemu rozumowaniu, którego praktycznym odzwierciedleniem jest np. OS/360, należy przeciwstawić inne: użytkownik powinien otrzymać minimalny komplet narzędzi, za pomocą których bez niczyjego pośrednictwa zbuduje sobie sam taki system operacyjny, jaki mu najbardziej odpowiada.

Osiągnięcie tego stanu, który praktycznie będzie oznaczał likwidację zawodów projektanta i programisty systemów operacyjnych nie jest realne w bliskiej przyszłości. Dążeniem do tego celu jest szukanie metod i narzędzi automatyzujących proces wytwarzania oprogramowania (a więc i systemów operacyjnych).

W dalszej części opracowania chcemy poświęcić nieco uwagi takim właśnie metodom i narzędziom.

2. BŁĘDY LOGICZNE A MOŻLIWOŚCI ZMIAN

System operacyjny może być rozpatrywany jako ta część systemu liczącego, która dzieląc instalację sprzętową pomiędzy niezależnych użytkowników odpowiada za najlepsze wykorzystanie systemu jako całości i jednocześnie za najlepszą obsługę każdego użytkownika. Centralna pozycja jaką zajmuje system operacyjny w systemie liczącym sprawia, że błędy popełnione przy jego projektowaniu lub produkcji rzutują na możliwości użytkowania całego systemu, poważnie komplikując lub wręcz uniemożliwiając korzystanie z niego (Pyle w [25] str. 71).

System operacyjny stanowi tę część oprogramowania, która musi być wykonana w pierwszej kolejności stając się podstawą całego oprogramowania. Nabudowywane na systemie operacyjnym oprogramowanie będzie ponosiło konsekwencje błędów popełnionych przy wykonywaniu systemu operacyjnego. Z kolei poprawienie błędów w systemie operacyjnym staje się praktycznie niemożliwe, gdyż oprogramowanie zaakceptowało te błędy i wszelkie dalsze usprawnienia systemu operacyjnego muszą starannie te błędy przenosić z edycji do edycji. Projektanci systemów operacyjnych lub translatorów języków programowania, które doczekały się maszynowej realizacji i były eksploatowane, doskonale znają tę koszmarną sytuację.

Projekt systemu operacyjnego w stopniu znacznie wyższym niż projekt jakiejkolwiek innej części oprogramowania musi uwzględniać możliwość nieustannych zmian w systemie w czasie jego użytkowania. Zmiany te dotyczą zarówno dostosowywania systemu do nowych potrzeb użytkowników (których nie można było przewidzieć) jak i do nowych możliwości sprzętowych (użytkownik będzie z reguły rozwijał system dołączając nowe urządzenia wnętrzne, powiększając pojemność pamięci itp.). W skrajnym przypadku, ale i z nim trzeba się liczyć, użytkownik może chcieć dokonać pełnej wymiany sprzętu.

System operacyjny, w którego projekcie nie przewidziano takich możliwości bądź przewidywano ale zagubiono w procesie realizacji, można uznać za stracony w przypadku konieczności

zmian. Wprowadzać zmiany do systemu operacyjnego, zawierającego w sobie błędy logiczne, jeśli ten system na dodatek posiada fatalną dokumentację, mogą w najlepszym przypadku tylko jego twórcy. Nikt, poza twórcami, nie jest w stanie tak zmodyfikować systemu operacyjnego, żeby wprowadzając doń nowe właściwości nie zaprzepaścić dotychczasowych (a więc zachować całe oprogramowanie). Autorzy referatu na własnym przykładzie odczuli, co to znaczy modyfikować system operacyjny będąc zarówno jego autorami (przejście z systemu S041 do S0141 na maszynie ZAM41) jak i będąc w pozycji użytkownika systemu (badania możliwości zmian w Executive maszyn ODRA System 1300). Oczywiście wiele zależy od zakresu zmian. Jednak uważamy za mało prawdopodobne, aby można było wykazać, że zmiany wykonane w istniejącym systemie nie powodują nieprzewidzianych błędów. Praktyka potwierdza nasze zdanie. Jako koronny przykład może służyć OS/360, którego logika uważana jest za niepoprawną, w wyniku czego często dostarczane użytkownikom nowe edycje systemu zawsze są obarczone dużą liczbą błędów (Pyle w [25] str. 71).

3. ELASTYCZNOŚĆ

Możliwość wykonywania zmian w systemie operacyjnym bądź w dowolnym programie nazywana bywa elastycznością (flexibility). Mówimy, że system operacyjny (lub dowolny program) jest elastyczny, jeśli jest przenośny (portable) i przystosowalny (adaptable).

Każdy program jest zaprojektowany do działania w jakimś otoczeniu (environment). Otoczeniem dla systemu operacyjnego jest system sprzętowy. Mówimy, że program jest przenośny, jeśli ma taką własność, że może być odwzorowany na różne otoczenia.

Program jest przystosowalny, jeśli umożliwia dostosowanie go do potrzeb użytkownika bez zmiany otoczenia.

Przenośność programu bywa określana jako miara łatwości z jaką program może być przenoszony z jednego otoczenia do drugiego: mówimy, że program jest w wysokim stopniu przenośny,

jeśli ilość pracy (koszt) włożona w przeniesienie programu jest znacznie mniejsza niż praca włożona w wykonanie go od początku. Podobnie można określić przystosowalność.

Główna różnica między przystosowalnością a przenośnością polega na tym, że przystosowalność jest związana ze zmianami struktury algorytmu, podczas gdy przenośność jest związana ze zmianami otoczenia [3, 18, 26].

W wielu publikacjach spotyka się określenie niezależności od maszyny (machine independence), które bywa traktowane bądź jako synonim przenośności bądź jako określenie własności słabszej (wg Goosa [9] wymaga się by program był niezależny od takich detali struktury maszyny jak długość słowa, liczba i rodzaj rejestrów itp.).

Bywa też stosowane pojęcie rozszerzalności (extendability) używane w sensie zbliżonym do elastyczności (np. [11]), ale z reguły odnoszone do różnych konfiguracji tego samego systemu sprzętowego (a więc bez wymagania niezależności od maszyny).

Przy okazji warto zauważyć, że jakkolwiek stopień trudności w uzyskiwaniu elastycznych systemów operacyjnych jest powszechnie dostrzegany, o tyle w wielu publikacjach wyraża się przypuszczenia, że uzyskanie systemów operacyjnych w dużym stopniu niezależnych od konkretnej maszyny (a więc potencjalnie przenośnych) jest prawdopodobne, gdyż stosunkowo niewiele żądań stawianych systemowi dotyczy bezpośrednio sprzętu, natomiast większość sprowadza się do zwykłego przetwarzania liczb, list itp. Szacuje się, że w większości współczesnych systemów operacyjnych, 60-70% objętości składających się na nie programów, mierzonej w instrukcjach maszynowych jest w istocie niezależna od maszyny (por. [3] i Hendry w [25] str. 64).

Zanim zajmiemy się rozważaniami na temat sposobu w jaki można uzyskać system operacyjny w możliwie wysokim stopniu elastyczny, spróbujmy określić jakie szczególne cechy różnią go od innych programów.

Po pierwsze: system operacyjny z założenia jako jedyny składnik oprogramowania m u s i być związany z konkretną maszyną (konieczność obsługi przerwań, fizycznych procesorów i pamięci itp.).

Po drugie: system operacyjny jako jedyny składnik oprogramowania m u s i być programem obliczenia współbieżnego [2] (współbieżne procesy w samym systemie operacyjnym i programy niezależnych użytkowników traktowane również jako procesy współbieżne).

Po trzecie: system operacyjny m u s i działać na programach. Prowadząc gospodarkę pamięcią wewnętrzną i zarządzając fizycznymi procesorami przeprowadza operacje ładowania programów do pamięci wewnętrznej, ich usuwania (są wtedy danymi) oraz ich wykonywania (przydziału fizycznego procesora - są wtedy programami).

Wymienione wyżej cechy mogą mieć także inne programy, podkreślamy jednak, że dla systemu operacyjnego są one konieczne.

Proces wytwarzania każdego programu obejmuje jego zaprojektowanie i wyprodukowanie (realizację maszynową - implementację). Obsługi technicznej (konserwacji) [26], wchodzącej również w skład procesu wytwarzania, nie będziemy tu omawiać. Projektowanie i maszynowa realizacja stanowią zwykle cykl iteracyjny i granica między nimi jest często trudno uchwytna, ale dla jasności rozważań będziemy mówili o projektowaniu i o produkcji jako o odrębnych czynnościach (choćby w tym rozumieniu, że mogą je wykonywać różni ludzie).

Przez zaprojektowanie programu będziemy rozumieli sporządzenie opisu wystarczającego do tego, żeby można było zarówno projektować inne programy z niego korzystające jak i wyprodukować ten program.

4. JĘZYKI PROJEKTOWANIA I PROGRAMOWANIA

Związki procesu wytwarzania systemów operacyjnych z językami programowania są oczywiste.

Najbardziej perspektywiczne wydaje się stosowanie języków wysokiego poziomu zarówno przy projektowaniu jak i przy realizacji maszynowej. Najlepiej byłoby, żeby język projektowania i język realizacji były tym samym językiem. O trudnościach jakie pojawiają się przy definiowaniu takiego języka mówi J. Olszewski [15].

Na razie nie ogłoszono, że taki język został określony. Są doniesienia, niestety bardzo lakoniczne, że systemy operacyjne dla maszyn Burroughs (B5500, B6500) są pisane w ALGOL-u 60 z rozszerzeniami (Cleary w [25] str. 67, Pyle w [25] str. 71, [21] i [5]) . Nie wiadomo, jakie szczególne warunki spełniają jednocześnie język, system operacyjny i maszyny, które były projektowane jako całość zarówno przez konstruktorów sprzętu jak i przez konstruktorów oprogramowania.

Spotkać się można z 3 zasadniczymi podejściami do stosowania języków programowania do wytwarzania systemów operacyjnych:

1) Wykorzystanie uniwersalnych języków wysokiego poziomu. Podejmuje się próby choćby częściowego wykorzystania przy wytwarzaniu systemów operacyjnych takich języków jak ALGOL 60 i różne jego rozszerzenia, ALGOL 68, PL/1 itp. [9, 18]. Przy budowie małego systemu operacyjnego (OS6 dla maszyny Modular One [24]) wykorzystano język BCPL [20] do zaprogramowania wszystkiego poza "małym jądrem". Pomijając już fakt, że system operacyjny OS6 jest jednoprogramowy nadmienić warto, że język BCPL nie narzuca konieczności pisania czytelnych programów. Prawdopodobnie taki system jest bardziej przenośny niż rozszerzalny czy przystosowalny do użytkownika. Uniwersalne języki wysokiego poziomu ułatwiają uzyskiwanie przenośnych programów jednak jej nie gwarantują automatycznie. W programach należy oddzielić wyraźnie strukturę algorytmów od struktur danych, gdyż głównie te ostatnie muszą być dopasowywane do nowego sprzętu.

Niektóre uniwersalne języki wysokiego poziomu (np. ALGOL 68) pozwalają zapisywać programy obliczeń współbieżnych, natomiast nie radzą sobie z funkcjami związanymi bezpośrednio z maszyną (byłoby to sprzeczne z ich uniwersalnością). Aby sprostać takim potrzebom proponuje się niekiedy wykorzystywanie języków wysokiego poziomu dopuszczających wstawki w językach symbolicznych.

2) Wykorzystywanie języków wysokiego poziomu zbliżonych do maszynowych. Takim językiem jest np. PL/360. [28]. Języki te umożliwiają m.in. używanie instrukcji maszynowych zapisywanych w formie wywoływania procedury. Jednak fakt, że zawierają np. pojęcia rejestrów rozumianych tak samo jak w konkretnej maszynie (dla PL/360 taką maszyną jest IBM 360), stawia pod znakiem zapytania ich przydatność z punktu widzenia przenośności. Do tej grupy można zaliczyć również makrogeneratory, które umożliwiają precyzowanie pojęć (ogólnikowo określonych na pewnym poziomie)na kolejnych poziomach niższych w ten sposób, że traktuje się je jako nazwy makrorozkazów, których definicje również zawierają makrorozkazy niższego poziomu itd, aż do zejścia na poziom kodu maszynowego [4]. W tej grupie języków najczęściej można spotkać języki - efemerydy, zaprojektowane tylko dla jednego przedsięwzięcia i stosowane jedynie przez autorów (por. Buxton w [25] str. 72). Języki takie, rzadko dobrze zaprojektowane, przynoszą więcej szkody niż pożytku, gdyż tworząc je zapomniano o tym, że język programowania co najmniej w takim stopniu jak do porozumiewania się z maszyną służy do porozumiewania się między ludźmi, umożliwiając czytanie i rozumienie napisanych przez kogoś programów. Uczyć się takiego języka tylko po to, aby zrozumieć jeden program jest oczywistym nonsensem.

3) Najbardziej powszechne - wykorzystywanie języków symbolicznych (assemblerów) . Najczęściej wysuwanym argumentem na ich korzyść jest fakt, że zapewniają najlepsze dopasowanie do konkretnego sprzętu i uzyskanie najbardziej sprawnych programów. Goos [9] trafnie zauważa, że program napisany

w języku symbolicznym może mieć wszystkie pożądane właściwości poza przenośnością. Dodajmy, że również czytelność programów napisanych w językach symbolicznych jest minimalna. Warto tu jednak zauważyć, że dysponując jedynie językiem symbolicznym można zastosować wspomniane wyżej podejście, uważane za charakterystyczne dla makrogeneratorów. Autorzy referatu przy projektowaniu struktury standardowego systemu operacyjnego SOS/1300 dla maszyn ODRA - System 1300 [14] definiowali pojęcia wyższego poziomu przy użyciu pojęć poziomu niższego itd., z założeniem, że dopiero w fazie kodowania (w języku symbolicznym) będzie podjęta decyzja, które definicje mają stanowić ciała procedur, a które będą bezpośrednio rozwinięte w ciele programu (lub procedury wyższego poziomu).

Problem działania na programach pozytywnie rozwiązują tylko języki symboliczne lub języki zbliżone do maszynowych. Języki uniwersalne wysokiego poziomu tej możliwości nie dają. Wyjątek stanowią języki konwersacyjne ale nie bardzo wiadomo jak do programowania systemów operacyjnych można je wykorzystać.

Tak więc z punktu widzenia przenośności (a więc elastyczności) najbardziej korzystnie przedstawiają się uniwersalne języki programowania wysokiego poziomu.

Często wysuwa się zarzut, że postulując przenośność systemów operacyjnych i w ogóle programów, mijamy się z rzeczywistymi potrzebami użytkowników. Użytkownik przenosząc program do nowego otoczenia będzie chciał go przy okazji poprawić. Możliwe. Jednak po pierwsze - zanim go poprawi - może go mieć w nowym otoczeniu w wersji niepoprawionej i tymczasowo z niego korzystać, po drugie - program zaprojektowany z myślą o przenośności o wiele łatwiej jest poprawiać, zwłaszcza jeśli zaprogramowano go w języku wysokiego poziomu.

Sprawność przenoszonego oprogramowania jest oczywiście gorsza niż projektowanego od nowa dla konkretnej instalacji, ale zawsze istnieje możliwość dostrojenia. Proponuje się np. żeby wykorzystywać w tym celu translatory języków programowania wysokiego poziomu produkujące programy wynikowe w języku symbo-

licznym. Takie programy można najdokładniej dopasowywać do konkretnego sprzętu i wymagań.

Kończąc uwagi na temat podejścia językowego do konstrukcji elastycznych systemów operacyjnych warto odnotować propozycje zalecające, żeby przy opracowywaniu języka wysokiego poziomu przeznaczonego do programowania systemów operacyjnych nie dążyć do uzyskania jednego języka, a raczej wprowadzić język z wieloma podjęzykami (oczywiście wysokiego poziomu), umożliwiającymi pisanie różnych fragmentów systemu operacyjnego w różnych językach [26]. Zwracamy tu uwagę na to podejście, bo właśnie programowanie modularne, o którym będzie dalej mowa, umożliwia spełnienie tego postulatu.

5. STRUKTURA HIERARCHICZNA

System operacyjny przekształca maszynę fizyczną w wirtualną. Program przekształcania maszyny fizycznej w wirtualną jest programem pisanym w języku tej fizycznej maszyny.

Wynika stąd, że chcąc wprowadzić drobną zmianę, która wymaga tylko małego stopnia wiedzy o sprzęcie, musimy często poznawać całą maszynę fizyczną. Maszyny zaś z reguły są opisywane bardzo nieprecyzyjnie, przeważnie w języku naturalnym i za pomocą pojęć, które zwłaszcza z punktu widzenia człowieka przyzwyczajonego do języka programowania wysokiego poziomu są egzotyczne: rejestr, przerwanie, obsługa kanału itp.

Jednopoziomowy, jednowarstwowy system operacyjny jest bardzo mało elastyczny, tzn. że nie bardzo wiadomo jak wiele trzeba w nim zmienić, żeby go przenieść z otoczenia do otoczenia, lub zachowując otoczenie zmienić definicję maszyny użytkownika.

Obraz zmienia się, gdy podzielimy system operacyjny na warstwy (layers) tak, żeby warstwa poziomu n określała maszynę wirtualną dla poziomu n+1 i wyłącznie za pomocą środków dostarczonych przez warstwę n-1. Warstwy realizują przekształcenie maszyny fizycznej w wirtualną stopniowo. Taką strukturę systemu nazywamy hierarchiczną.

Uzyskujemy więc system podzielony na części składowe. Podział systemu na części pozwala osiągnąć dwa główne cele:

1. możliwość uzasadnienia poprawności logicznej (i znalezienia błędów) na etapie projektowania,

2. możliwość uzyskania systemu operacyjnego podatnego na zmiany, elastycznego.

Wyżej mówiliśmy o potrzebie zarówno poprawności jak i elastyczności. Zastanówmy się jaki związek te pojęcia mogą mieć z podziałem systemu operacyjnego na warstwy. Nie będziemy tu rozważać w jaki sposób można przeprowadzić dowód poprawności programu (a więc i systemu operacyjnego), jest to osobne, obszerne zagadnienie. Warto tu nadmienić, że mówiąc o poprawności programu właściwie mamy na myśli weryfikację naszego sądu o nim, gdyż naprawdę nie istnieją programy niepoprawne - istnieją tylko programy inne niż sobie wyobrażamy (A. Mazurkiewicz: Problemy programowania*).

Parnas [17] podkreśla, że dowody poprawności programów mogą okazać się tak złożone, że poprawność tych dowodów stoi pod znakiem zapytania. Im prostszy program, tym dowód jego poprawności będzie prostszy. Jest to argument za dzieleniem programu na części składowe, których poprawność można łatwo udowodnić i z kolei wykorzystać te dowody przy dowodzeniu poprawności całego programu.

Związek między podziałem systemu operacyjnego na części a możliwością zmian też widać wyraźnie. Można uzyskać taki podział systemu, żeby w celu wprowadzenia zmian przeprojektować tylko nieliczne jego części, a pozostałe zostawić nienaruszone. Zmiany mogą więc mieć lokalny charakter.

Często wyżej niż poprawność stawia się niezawodność bądź efektywność (przy czym zwłaszcza efektywność bywa bardzo róż-

---

* Materiały na sesję naukową z okazji Roku Nauki Polskiej i XV-lecia IMM, zesz. 1, Warszawa 1973, str. 1-29.

nie i często nieprecyzyjnie rozumiana). Łatwo jednak wykazać, że logicznie poprawny system można uczynić niezawodnym i efektywnym jak również, że nie można tego dokonać w stosunku do systemu z logicznymi błędami. Z kolei nieuchronne modyfikacje systemu, w którym nie przewidziano możliwości zmian mogą się odbywać tylko kosztem utraty zarówno niezawodności jak i efektywności.

Zastanówmy się teraz jak przeprowadzić taki podział systemu operacyjnego na warstwy.

Jeśli przyjąć za warstwę najniższą tę, która bezpośrednio styka się ze sprzętem (stanowi oprogramowanie tego sprzętu), a za najwyższą - warstwę obejmującą programy użytkowników (lub samego użytkownika w wariancie konwersacyjnym), to każda warstwa opisywana jest w kategoriach funkcji realizowanych przez warstwy niższe. Czyli można powiedzieć, że poszczególne warstwy wprowadzają usprawnienia programowe wykorzystywane w warstwach wyższych [26].

Przykładem tak zaprojektowanego systemu jest system operacyjny "THE" [8], który zrealizował Dijkstra na maszynie Electrologica X8. Ściśle hierarchiczna struktura jaką przyjmuje system jest zaprojektowana głównie z myślą o tym, żeby łatwo było udowodnić poprawność systemu.

Rozważa się sześć warstw (poziomów):

Warstwa 0 - w tej najniższej warstwie w powiązaniu z obsługą przerwań zegarowych realizowany jest przydział fizycznego procesora jednemu z procesów realizowanych w warstwach wyższych. Powyżej tego poziomu nie istnieją procesory fizyczne.

Warstwa 1 - przekształca dwupoziomową pamięć fizyczną (wewnętrzna i bębnowa) na pamięć jednopoziomową. Powyżej tego poziomu operuje się wyłącznie pamięcią wirtualną.

Warstwa 2 - przekształca fizyczną konsolę (monitor dalekopisowy) w niezależne konsole konwersacyjne przypisane do każdego z procesów.

Warstwa 3 - przekształca fizyczne urządzenia wejścia i wyjścia w strumienie wejściowe i wyjściowe. Liczba fizycznych urządzeń przestaje być istotna na poziomach wyższych.

Warstwa 4 - obejmuje procesy niezależnych użytkowników.

Warstwę 5 - stanowi operator.

Przyjęta struktura umożliwia operowanie pojęciem procesu sekwencyjnego, począwszy od poziomu 1, umożliwia też lokalizowanie w obrębie warstwy ewentualnych modyfikacji sprzętowych. Np. warto zauważyć, że zmiana liczby fizycznych procesorów będzie wymagała przebudowy jedynie warstwy 0 [7]. Według podobnych zasad konstruowany był również system operacyjny RC 4000 [2, 19].

Liczba warstw w systemach hierarchicznych nie koniecznie musi być taka jak u Dijkstry. Często przekształca się w wirtualne zarówno fizyczne procesory jak i pamięci w obrębie tej samej warstwy - mówi się wtedy z reguły o "bardzo małym jądrze" napisanym w kodzie maszyny.

Przykładem takiego systemu może być system operacyjny MU5 [12, 13] realizowany na Uniwersytecie w Manchester dla zestawu maszyn (na obecnym etapie: maszyny MU5 i ICL 1905E). W systemie tym tylko warstwa najniższa (nazywana tu Supervisor Supervisor) zrealizowana jest na maszynie fizycznej. Cała reszta stanowi oddzielne programy nazywane Supervisorami działające już na poziomie wyższym (korzystające z własnych maszyn wirtualnych) i definiujące nowe maszyny wirtualne dla różnych klas użytkowników (będą to np. Supervisory: Batch Job Control, Interactive Job Control, Reservation System itp.). Programy użytkowników znajdują się oczywiście na jeszcze wyższym poziomie.

W przypadku struktury hierarchicznej istotne jest aby dana warstwa mogła być opisana wyłącznie przy użyciu pojęć warstwy

bezpośrednio niższej - a nie dalszych. Oczywiście niektóre pojęcia przenoszone są przez poszczególne warstwy bez zmian, ale godząc się na koszt realizacji takich jałowych przejść możemy się ustrzec od przykrych błędów, które powstają przy konieczności sięgania do warstw dalszych.

Taka konieczność pojawia się w systemach zaprojektowanych co prawda warstwowo, lecz nie konsekwentnie. Istnieją takie systemy operacyjne gdzie (jak np. w SO 141 maszyny ZAM 41) w tej samej warstwie korzysta się z różnych poziomów abstrakcji dotyczących tych samych urządzeń zewnętrznych - powstaje wtedy możliwość konfliktów, trzeba budować sztuczne zabezpieczenia itd.

Chcielibyśmy wyrazić przekonanie, że systemów bez wyraźnej, warstwowej struktury nie da się dobrze opisać. Nie można bowiem uznać za dobry opis czegoś, co jest w istocie katalogiem właściwości z różnych poziomów - obszernym i nieczytelnym.

Projektowanie i programowanie systemu o strukturze hierarchicznej najlepiej byłoby przeprowadzić przy użyciu języka proceduralnego wysokiego poziomu. Jak wiadomo są kłopoty z uzyskaniem takiego języka przydatnego do programowania systemów operacyjnych. Wszelkie przybliżenia są jednak racjonalne. Toteż sposób realizacji maszynowej danej warstwy można uznać za obojętny byleby uzyskana na styku z warstwą wyższą maszyna wirtualna rzeczywiście miała założone właściwości. Można mówić o dwóch rodzajach opisu warstwy: dla użytkownika - czyli definicja maszyny danego poziomu wyrażona w pojęciach tego samego poziomu oraz dla wykonawcy - definicja maszyny danego poziomu wyrażona w pojęciach poziomu bezpośrednio niższego.

6. MODULARNOŚĆ

Modularność można traktować jako uogólnienie warstwowości. Podział systemu można prowadzić warstwowo (modularyzacja pionowa) albo nie dzieląc systemu na warstwy wyodrębnić w nim części o różnych funkcjonalnie właściwościach (modularyzacja

pozioma). Praktycznie - zwłaszcza przy dużych systemach podział jest prowadzony i w pionie i w poziomie, tzn. warstwa dzielona jest na moduły.

Definicja modułu zależy zarówno od tego, po co się dzieli, jak i od tego, jak się opisuje strukturę. Parnas [17] określa moduł po prostu jako część wskazaną przez opis systemu.

Potrzebę podziału systemu na warstwy wykazaliśmy - warto również zauważyć, że podział warstwy na mniejsze moduły często pozwala zastąpić potrzebę przeprogramowywania całej warstwy wymianą jednego modułu.

Rozmiar modułu ustala się zwykle na wyczucie - bo kryteria są sprzeczne: im więcej modułów - tym łatwiejsze zmiany, ale z kolei tym więcej powiązań między nimi, co może oznaczać potrzebę większej liczby przesłań informacji np. między warstwami.

Moduł jako jednostka projektowania musi być opisany. Opis modułu ogólnie też trudno wyrazić. O ile w przypadku warstwy rzecz jest pojęciowo prosta, o tyle w innych podziałach trzeba sformułować ostrożniejsze postulaty. Potrzebne są dwa opisy modułu: jak z niego korzystać i jak go zbudować. Zdaniem Parnasa [16] opis modułu powinien zawierać absolutne minimum wiedzy, wystarczające do tego, aby moduł zbudować i aby z niego skorzystać. W szczególności opis modułu nie powinien podawać jego struktury wewnętrznej.

Często uważa się, że moduły mogą być opisywane w języku naturalnym. Podejście takie nie wydaje się słuszne. Według Parnasa [16] opis modułu powinien być na tyle sformalizowany, żeby można go było poddać maszynowemu testowaniu (np. czy opisuje wszystkie możliwe stany modułu), a w każdym razie przeprowadzić rozumowanie pozwalające na podstawie samych opisów wykazać poprawność logiczną systemu.

Niekiedy modularność rozważa się jedynie na etapie projektu. Tymczasem istotną zaletą modularności jest, że części systemu wyróżnione w projekcie zachowują tożsamość w gotowym systemie.

W konsekwencji dokumentacja projektowa - sformalizowany opis modułu przez cały czas działania systemu może pozostać jedynym dokumentem. Czynnikiem ułatwiającym opis i korzystanie jest standaryzacja, a więc traktowanie modułu jako pewnego rodzaju pojemnika (być może jest kilka typów pojemników różniących się kształtem), który można w typowy sposób zestawiać z innymi.

Rozważmy przykładowo, że moduł miałby kształt procedury np. w sensie PASCAL-a [29]. Jeśliby wśród parametrów wywołania istniał ustalony parametr typu boolean wskazujący np. czy zakończyć proces (terminate process[2]) czy też nie, to ten sam moduł mógłby być w zależności od sposobu wywołania albo zwykłą procedurą albo programem procesu sekwencyjnego.

Opis systemu operacyjnego sporządzony w języku wysokiego poziomu byłby opisem jego poszczególnych warstw zrealizowanych w formie procedur, przy czym, im głębiej zanurzona byłaby deklaracja procedury - tym bliżej sprzętu leżącą warstwę opisywałaby.

Moduł może być traktowany jako operacja obliczenia (współbieżnego lub sekwencyjnego). Z kolei modułowi odpowiada obszar pamięci wymagany do jego działania. Będąc operacją (a więc i obliczeniem) moduł pozostaje jednocześnie segmentem danych, które można przenosić z miejsca na miejsce (co spełnia wymaganie by można było traktować programy jako dane i odwrotnie). Uzyskujemy w ten sposób właściwość dostępną jedynie na poziomie języka symbolicznego (zawartość każdego miejsca pamięci - tu: moduł - może być traktowana zarówno jako operacja jak i jako dana).

Konsekwentnie wprowadzając modularność można uzyskać system podatny na zmiany. Przy czym w znacznym zakresie zmienność tę uzyskuje się przez tworzenie bibliotek modułów pogrupowanych warstwami, z których można składać różne systemy dla różnych potrzeb użytkowania (warstwy wyższe) oraz dla różnych rozszerzeń i być może głębokich zmian sprzętu (warstwy niższe). Można również mając moduły w różnych wykonaniach (np. krótsze ale wolniejsze, dłuższe ale szybsze) uzyskiwać systemy różne w sensie efektywności czy niezawodności.

Realizacja maszynowa modularnego systemu operacyjnego może przebiegać w ten sposób, że moduły będą pisane i testowane (ze względu na zgodność z opisem) w dowolnej kolejności, a więc w szczególności na raz przez większą grupę programistów. Dopiero kompletowanie systemu musi przebiegać warstwami - od najniższej poczynając.

Skoro struktura wewnętrzna modułu nie jest precyzowana w jego opisie, istnieje dość duża swoboda w stosowaniu środków programowych przy jego wykonywaniu. Różne moduły można zaprogramować wykorzystując różne języki programowania.

Zastosowanie języka wysokiego poziomu do wyprodukowania modułu automatyzuje jego produkcję dla różnych systemów sprzętowych.

Na końcu zostaje do omówienia problem warstwy najbliższej sprzętu (definiującej proces). Można tylko wyrazić życzenie by była jak najmniejsza. Nie znamy żadnego zadowalającego opisu struktury warstwy 0. Musi być wyrażona za pomocą pojęć maszyny fizycznej, a te dla różnych maszyn są bardzo różne.

Wydaje nam się, że przynajmniej ta warstwa musi być zrealizowana od nowa przy przejściu z maszyny na maszynę. Produkcję tej warstwy można zautomatyzować jedynie przy pomocy projektantów sprzętu, jak to ma miejsce np. u Burroughsa [5].

7. WNIOSKI KOŃCOWE

Staraliśmy się wykazać, że jakkolwiek brak języka wysokiego poziomu do pisania systemów operacyjnych jest przeszkodą w uzyskaniu pełnej automatyzacji wytwarzania systemów operacyjnych w ogóle a elastycznych w szczególności, to możliwe są rozwiązania częściowe, pozwalające uzyskiwać pożądane efekty nawet przy braku takiego języka.

Najbardziej obiecującym podejściem jest nadanie systemowi operacyjnemu na etapie projektowania konsekwentnej hierarchicznej struktury warstwowej, umożliwiającej zarówno wczesne wykry-

cie błędów projektu, jak i szerokie modyfikacje zarówno w zakresie sprzętu (warstwy niższe) jak i w zakresie sposobów użytkowania (warstwy wyższe).

Dzieląc warstwy na dające się funkcjonalnie wyodrębnić fragmenty uzyskujemy możliwość łatwiejszego modyfikowania warstwy. Opisując warstwy oraz ich elementy (moduły) w sposób możliwie formalny uzyskujemy już na etapie projektowania dokumentację systemu wystarczająco precyzyjną zarówno do tego, by elementy te wykonać, jak i do tego, by móc się nimi posługiwać.

Poszczególne moduły można odwzorowywać w konkretnym systemie sprzętowym za pośrednictwem różnych dostępnych i najwłaściwszych narzędzi programowych (w tym również języków wysokiego poziomu), mogą być wykonywane równocześnie przez różnych programistów, co przyspiesza wykonanie całości systemu.

Z gotowych modułów można szybko składać różne wersje systemu operacyjnego, który można usprawniać (podnosić efektywność) przez wymianę poszczególnych modułów na nowe, specjalnie starannie zaprogramowane, co jednak wobec lokalności zmian odbywać się może bez większych zakłóceń w działaniu systemu.

Należy sądzić, że jeśli moduły warstw pośrednich będą zaprogramowane w języku wysokiego poziomu, to nawet przy radykalnych zmianach sprzętu (przeprogramowanie warstw niższych) oraz daleko idących zmianach sposobów użytkowania (przeprogramowanie warstw wyższych) będą mogły być wykorzystane w zmienionym systemie.

Podejście to umożliwia praktyczne przygotowanie oprogramowania dla przyszłych systemów liczących, o których sprzęcie oraz sposobach użytkowania ma się tylko bardzo ogólnikowe wyobrażenia.

A więc nie znając uniwersalnego sposobu rozwiązywania tak złożonego problemu jakim jest automatyczne wytwarzanie systemów operacyjnych, znajdujemy praktyczne przybliżenie uzyskane przez podział procesu wytwarzania na odrębne etapy: projektu i maszynowej realizacji z jednej strony, a z drugiej - przez

podział samego produktu na części, z których każdą można wystarczająco precyzyjnie opisać i wystarczająco łatwo wykonać.

Okazuje się, że w ten sposób można wskazać również te problemy, których rozwiązania nie może zaniechać poszukiwany język programowania systemów operacyjnych, jeśli ma stać się tym jedynym narzędziem, pozwalającym użytkownikowi być sam na sam z maszyną i kształtować ją według swoich potrzeb i upodobań.

UWAGI DOTYCZĄCE LITERATURY

W opracowaniu wykorzystaliśmy wiele publikacji, z których nie wszystkie były cytowane wprost. Chcemy zwrócić uwagę na te, które naszym zdaniem są istotne.

Najszerzej zagadnienia wytwarzania oprogramowania potraktowane są w materiałach na temat inżynierii oprogramowania [22, 23]. Cenne uwagi na tematy związane bezpośrednio z problematyką opracowania zawiera praca W.M. Turskiego [26].

Literatura dotycząca wytwarzania elastycznych programów z reguły odnosi się do programów użytkowych, sekwencyjnych i mocno oddalonych od konkretnego sprzętu. Warto tu odnotować zwłaszcza pracę Poole´a i Waite´a [18] (zaopatrzoną zresztą w obszerną bibliografię) oraz Browna [3]. Natomiast problematyka elastycznych systemów operacyjnych nie jest zbyt szeroko przedstawiana w literaturze. Cox [4] formułuje wiele ogólnych uwag na temat niezależności systemu operacyjnego od sprzętu, Lampson [11] oraz Stoy i Strachey [24] przedstawiają swoje poglądy na temat budowy takich systemów.

Problematyką modularności zajmuje się wielu autorów, również głównie w odniesieniu do programów użytkowych. Można tu m.in. wskazać prace: Dennisa [6], Parnasa [16, 17], Judda [10]. Przykłady systemów operacyjnych modularnych można znaleźć w pracach Morrisa [12, 13], Wichmanna [27], Bertina [1].

Literatura

[1] BERTIN J., RITOUT M., ROUGIER J.C.: L'exploitation partage des calculateurs. Dunod, Paryż 1967 (tłum. ros. 1970).

[2] BRINCH-HANSEN P.: Operating Systems Principles, Prentice-Hall. 1973.

[3] BROWN W.S.: Software Portability, Software Engineering Techniques, Working Papers, Buxton J.N. and Randell B. (Eds.), NATO Science Comm., Rome 1969, 1970, s. 80-84.

[4] COX P.R.: Machine-independent Operating Systems: a Functional Approach to Design, The Fourth Generation, International Computer State of the Art Report, Presentations, INFOTECH 1971, s. 239-258.

[5] CREECH B.A.: Architecture of the B6500, Software Engineering, COINS III, Tou J.T. (Ed.), Academic Press 1970, t. 1, s.29-43.

[6] DENNIS J.B.: Modularity, Advanced Course on Software Engineering, Bauer F.L. (Ed.), Lecture Notes in Econom. and Math. Syst., 81, Springer-Verlag 1973, s. 128-182.

[7] DIJKSTRA E.W.: Structured Programming, Software Engineering Techniques, Working Papers, Buxton J.N. and Randell B. (Eds.), NATO Science Comm., Rome 1969, 1970, s. 84-88.

[8] DIJKSTRA E.W.: The Structure of the "THE" - Multiprogramming System, Comm. ACM, 1968, t. 11, nr 5, s. 341-346.

[9] GOOS G.: Programming Languages as a Tool in Writing System Software, Advanced Course on Software Engineering, Bauer F.L. (Ed.), Lecture Notes in Econom. and Math. Syst., 81, Springer-Verlag 1973, s. 47-69.

[10] JUDD R.: Practical Modular Programming, The Computer Bulletin, 1970, t. 14, nr 1, s. 4-7.

[11] LAMPSON B.W.: On Reliable and Extendable Operating Systems, The Fourth Generation, International Computer State of the Art Report, Invited Papers, INFOTECH 1971, s. 421-442.

[12] MORRIS D.: The Nature and Benefits of Modular Operating Systems, The Fourth Generation, International Computer State of the Art Report, Presentations, INFOTECH 1971, s. 279-298.

[13] MORRIS D., DETLEFSEN G.D.; FRANK G.R., SWEENEY T.J.: The Structure of the MU5 Operating System, Comp. J., 1972, t. 15, nr 2, s. 113-116.

[14] MYSIOR H., MYSIOR J.: Funkcje i struktura programu sterującego wielozadaniową pracą maszyn cyfrowych ODRA-System 1300 w czasie rzeczywistym, Materiały z sympozjum roboczego nt. projektowania systemów operacyjnych do pracy w czasie rzeczywistym, Politechnika Gdańska, Instytut Okrętowy, Gdańsk 1972, Publ.nr 36.

[15] OLSZEWSKI J.: Problemy strukturalnego programowania systemów operacyjnych, w niniejszym zeszycie.

[16] PARNAS D.L.: A Technique for Software Module Specification with Examples, Comm. ACM, 1972, t. 15, nr 5, s. 330-336.

[17] PARNAS D.L.: Information Distribution Aspects of Design Methodology, IFIP Congress 71, Booklet TA-3, 1971, s. 26-30.

[18] POOLE P.C., WAITE W.M.: Portability and Adaptability, Advanced Course on Software Engineering, Bauer F.L. (Ed.), Lecture Notes in Econom. and Math. Syst., 81, Springer-Verlag 1973, s. 183-277.

[19] RC4000 Software, Multiprogramming System, Brinch Hansen P. (Ed.), A/S Regnecentralen Kopenhaga 1969, RCSL no: 55-D17.

[20] RICHARDS M.: BCPL: a Tool for Compiler Writing and System Programming, Proc. SJCC, AFIPS 1969, s. 557-566.

[21] ROSEN S.: Programming System and Languages, Comm. ACM, 1972, t. 15, nr 7, s. 591-610.

[22] Software Engineering, Naur P. and Randell B. (Eds.), NATO Science Comm., Garmisch 1968, publ. 1969.

[23] Software Engineering Techniques, Buxton J.N. and Randell B. (Eds.), NATO Science Comm., Rome 1969, publ. 1970.

[24] STOY J., STRACHEY C.: OS6 - an Operating System for a Small Computer, The Computer Journal 1972, t. 15, nr 2, s. 117-124.

[25] The Fourth Generation, International Computer State of the Art Report, INFOTECH 1971.

[26] TURSKI W.M.: Projektowanie oprogramowania systemów liczących, Projektowanie maszyn i systemów cyfrowych, PWN 1972, s. 123-144.

[27] WICHMANN B.A.: A Modular Operating System, IFIP Congress 68, Booklet C, 1968, s. 48-54.

[28] WIRTH N.: PL/360, a Programming Language for the 360 Computers, Journal ACM, 1968, t. 15, s. 37-74.

[29] WIRTH N.: The Programming Language Pascal, Acta Informatica, 1971, t. 1, nr 1, s. 35-63.

МОДУЛЯРНОЕ ПРОГРАММИРОВАНИЕ ОПЕРАЦИОННЫХ СИСТЕМ

Резюме

В работе представляются проблемы проектирования и реализации гибких операционных систем. В проекте определяется иерархическая слоистая структура операционной системы, слои которой разделяются затем на меньшие части - модули. Для разработки модулей и их использования необходимо дать их формальное описание, а также описание способа их объединения в конкретную систему. Это описание является достаточной документацией операционной системы.

Внутреннее строение модулей не определяется. Модули можно программировать при использовании разных языков программирования.

MODULAR PROGRAMMING OF OPERATING SYSTEMS

Summary

This paper presents some problems of designing and implementation of flexible operating systems. In design stage hierarchical layered structure of an operating system is defined. Layers are divided into smaller items - modules. Formalized modules specifications are needed for their implementation and application as well as linkage specifications to construct full actual operating system. Such specifications mentioned above provide also sufficient system documentation. Internal structure of modules is not defined. They may be implemented by means of different programming languages.

Instytut Maszyn Matematycznych
02-078 Warszawa, ul. Krzywickiego 34

ALGORYTMY, Zeszyt Specjalny, 1974, 157-171 681.322.06:
061.3/438/"1973"

PROBLEMY STRUKTURALNEGO PROGRAMOWANIA SYSTEMÓW OPERACYJNYCH

Jacek OLSZEWSKI
Instytut Maszyn Matematycznych
Pracę złożono 10.I.1974

Treścią opracowania jest próba odpowiedzi na pytanie, dlaczego języki proceduralne w ich obecnej postaci, a nawet w postaci projektów specjalnie pomyślanych dla opisu poszczególnych części systemu operacyjnego, nie służą jako języki ich programowania.

S p i s t r e ś c i

1. WSTĘP

Od dawna próbowano i nadal próbuje się mierzyć wydajność pracy programisty liczbą rozkazów, z których układa on program, na jednostkę czasu. Nie liczy się przy tym rozkazów, które były błędne, skreślone, a tylko takie, które pozostały w programie po jego sprawdzeniu i uruchomieniu. Przy takim sposobie oceny pracy korzystnie wypadają ci, którym przypadło układać programy krótkie. Okazuje się bowiem, że wysiłek intelektualny włożony

w ułożenie, testowanie, poprawianie tzn. wszystko to, co się nazywa uruchamianiem programu, nie jest proporcjonalny do jego długości. Im dłuższy program, tym wydajność, nawet tego samego programisty, mierzona liczbą rozkazów na jednostkę czasu, jest mniejsza. W dziedzinie programowania systemów operacyjnych wydajność ta w miarę wzrostu programów spada jeszcze szybciej, niż w innych dziedzinach.

Mógłby ktoś zauważyć, że w takim razie należy długi program rozbić na kilka lub kilkanaście krótkich, a jeśli te krótkie też są za długie, to trzeba rozbijać dalej. Uwaga o tyle nie trafna, że nie ma sensu rozkładać na kawałki p r o g r a m u, i to sprawdzonego i "chodzącego". Programista dostaje przecież do rozwiązania p r o b l e m. Program jest rozwiązaniem tego problemu. Należy więc próbować rozbić na części problem, a nie jego rozwiązanie. Przeciwnie, programy, które będą rozwiązaniami problemów częściowych, należy potem złożyć w jeden, stanowiący rozwiązanie całego problemu.

Powyższe jest w sposób oczywisty słuszne w odniesieniu do każdej, bardziej skomplikowanej działalności, a nie tylko do programowania. Jednakże w dziedzinie programowania podejście takie zyskało nawet swoją nazwę, a mianowicie nazywa się programowaniem strukturalnym. Lecz nawet wtedy, gdy nazwa ta jeszcze nie była w użyciu, strukturalne podejście do programowania było cechą dobrego programisty* . Leżało również u podstaw dążeń w kierunku języków programowania tzw. wysokiego poziomu. Bo przecież zdania złożone, procedury czy funkcje były i są programami, stanowiącymi rozwiązania częściowych problemów, a złożenie takich części w jeden program było (i jest) dokonywane automatycznie przez kompilator danego języka. Już z tego widać, że język wysokiego poziomu jest istotną pomocą w strukturalnym programowaniu. Oczywiście, język taki nie narzuca struktury programu; można w każdym z nich układać także zgoła niestrukturalne programy. Struktura programu musi wyniknąć ze struktury problemu, a rolą programisty będzie ułożyć

---

* Materiały na Sesję Naukową z okazji Roku Nauki Polskiej i XV-lecia IMM, A. Mazurkiewicz - "Problemy programowania" (oprac. wewnętrzne IMM).

program, który tę ostatnią możliwie dokładnie odzwierciedli. To jednak nie wszystko, należy tego dokonać tak, aby każdej "podstrukturze" problemu odpowiadała odpowiednia "podstruktura" programu oraz by mogła ona być układana, sprawdzana i uruchamiana niezależnie od wszystkich tych samych czynności odnoszących się do innych "podstruktur" tego samego programu. Powyższy warunek nie może być spełniony bez posługiwania się językiem wysokiego poziomu. Bez niego bowiem trudno byłoby mówić o niezależnym traktowaniu "podstruktur" tego samego programu. W takim przypadku musielibyśmy z góry przewidywać wielkości poszczególnych części programu, ich rozplanowanie w pamięci, sposób przekazywania danych z jednej części do drugiej i wiele tym podobnych rzeczy.

Jednakże mając nawet do dyspozycji język wysokiego poziomu i jego kompilator nie możemy problemu uważać za rozwiązany. Program ułożony strukturalnie powinien być poprawny nie tylko w odniesieniu do reguł gramatycznych danego języka, lecz również spełniać szereg warunków wynikających z tego, co wyżej zostało powiedziane. A więc powinien mieć postać pewnej hierarchii programów (procedur, funkcji) tak skonstruowanej, by na każdym jej poziomie można było rozważać i w pełni rozumieć "co się w programie dzieje" bez konieczności rozważania przy tym "jak to się dzieje", czyli bez zagłębiania się w niższy poziom. A gdy na pewnym poziomie struktury programu chcemy rozważyć "jak się to dzieje", wtedy powinien wystarczyć tylko jeden "krok w dół", a więc rozważenie, "co się dzieje" na poziomie bezpośrednio niżej.

To, "co się dzieje" na jakimś poziomie, nie powinno również zależeć od tego, co jest na wyższych poziomach. Innymi słowy procedura lub funkcja powinna być sprawdzalna w dowolnym kontekście, tj. niezależnie od tego, przez którą inną będzie wykorzystywana.

Jeden z kierunków, w jakich następuje rozwój języków programowania, odpowiada dość ściśle dążeniom, o których była mowa wyżej. W wielkim skrócie kierunek ten można wyrazić jako

ALGOL-PASCAL-PEARL [5, 6, 7], przy czym PEARL (Program Elaboration and Refinement Language) jest na razie tylko projektem, jakkolwiek w dużej części zrealizowanym i sprawdzonym.

Z opisanym wyżej sposobem programowania związane są nadzieje zarówno na to, żeby wysiłek potrzebny do ułożenia programu był proporcjonalny do jego wielkości, jak też na to, by program stanowił swoją łatwo czytelną dokumentację.

W dziedzinie systemów operacyjnych sytuacja, i to w obu wymienionych aspektach, wygląda, jak wiemy, bardzo źle. Do zaprogramowania takich systemów potrzebni są najwyższej klasy specjaliści i całe lata ich pracy. Zaś napisane i uruchomione programy nie mogą być traktowane jako łatwo czytelna dokumentacja systemu. Wobec tego powstają oddzielne opracowania, które, jeśli stanowią pełną dokumentację, to są bardzo obszerne i trudno w nich cokolwiek znaleźć.

Zatem wydawałoby się, że programowanie strukturalne z wykorzystaniem języków wysokiego poziomu powinno mieć szczególne powodzenie w tej dziedzinie. Jednak tak nie jest, po pierwsze dlatego, że programy napisane w takich językach uważa się za nieefektywne w porównaniu z programami ułożonymi ręcznie, zaś system operacyjny ma być zbiorem programów możliwie najefektywniejszych (m.in. najszybszych), a po drugie, struktury systemów operacyjnych nie dają się w sposób bezpośredni i prosty odwzorować na odpowiednie struktury programów napisanych w obecnie używanych językach. Jakkolwiek na temat pierwszej grupy przyczyn takiego stanu rzeczy można wieść długie dyskusje - co jest lepsze: czy szybsze ułożenie niezbyt efektywnego programu, czy też bardzo mozolne układanie programu, o którym też nie wiadomo, czy będzie najefektywniejszym z możliwych - to druga grupa przyczyn stanowi wg autora, istotny problem wymagający przynajmniej prób rozwiązania.

Na niektóre aspekty tego problemu chcemy zwrócić uwagę w dalszym ciągu referatu. Do pełnego zrozumienia krótka uwaga na temat pojęcia deklaracji: otóż deklaracje będą rozumiane dwojako - w tekście programu są to odpowiednie jego kawałki określa-

jące zmienne, procedury i funkcje "ważne" w danym bloku (procedurze lub funkcji), natomiast podczas wykonywania programu będziemy mówić o dokonywaniu deklaracji lub deklarowaniu zmiennych, procedur i funkcji. W przypadku zmiennej będzie to pewna operacja na pamięci, polegająca na wydzieleniu odpowiedniej części pamięci oraz stowarzyszeniu z nią nazwy tej zmiennej. Należy oczywiście przyjąć, że wraz z zakończeniem wykonywania danego bloku (procedury lub funkcji) następuje operacja odwrotna czyli zwolnienie zajętej dotychczas części pamięci.

2. ZMIENNE LOKALNE I GLOBALNE

Z przedstawionej we wstępie ogólnej idei programowania strukturalnego wynikają pewne praktyczne wnioski odnośnie tego, jak układać programy - m.in. nie korzystać z możliwości deklarowania zmiennych globalnych, czy, innymi słowy, nie traktować żadnych zmiennych jako globalnych. Zamiast tego należy posługiwać się mechanizmem przekazywania parametrów - wówczas będzie spełniony warunek, by procedura lub funkcja była niezależna od tego "co się dzieje" na wyższym poziomie, a więc sprawdzalna w jakimkolwiek kontekście. Zauważmy, że jakkolwiek w ALGOL-u każda zmienna może być traktowana dwojako: lokalnie na tym poziomie, gdzie została zdeklarowana, i globalnie na poziomach niższych, to w PASCAL-u mamy już pewne ograniczenie, co do korzystania ze zmiennych globalnych; podstawianie na zmienną globalną jest traktowane jako błąd formalny i wykrywane podczas tłumaczenia programu. W ten sposób unikamy tzw. efektów ubocznych działania procedur lub funkcji. Natomiast w PEARL-u w ogóle pojęcie zmiennej globalnej nie istnieje; wszystkie zmienne są lokalne, a jeżeli mają być dostępne na niższych poziomach, to tylko jako parametry.

Układając program w którymkolwiek z trzech wymienionych języków należy wiedzieć, że podczas wykonywania tego programu zmienne będą "powoływane do życia" w momencie ich deklaracji, co następuje przy rozpoczęciu wykonywania odpowiedniego bloku (w PASCAL-u - procedury lub funkcji, w PEARL-u - operacji), na-

tomiast "kończą swój żywot", gdy kończy się wykonywanie bloku, w którym zostały zadeklarowane. To samo odnosi się do związków pomiędzy parametrami formalnymi i aktualnymi procedur lub funkcji. Określenie tych związków następuje przy wywołaniu procedury lub funkcji, natomiast gdy wykonywanie jej kończy się, wtedy "ustaje" również jakikolwiek związek pomiędzy parametrami formalnymi i aktualnymi.

Z drugiej strony wiadomo, że w systemie operacyjnym mamy procedury działające na zmiennych, których "czas życia" jest znacznie dłuższy niż ich wykonywanie, a ponadto zmienne te nie powinny być dostępne nigdzie poza tymi procedurami. Zatem byłoby błędem zarówno umieszczać ich deklaracje w ciałach owych procedur, czyli traktować je jako lokalne, jak i deklarować je na jakimś wyższym poziomie, a w owych procedurach działać na nich w postaci parametrów. Wtedy bowiem byłyby one dostępne również poza tymi procedurami.

Język ALGOL 60 w swoim wydaniu wzorcowym (patrz Naur [5]) umożliwia co prawda rozwiązanie powyższego problemu w sposób bardzo prosty. Mamy w tym języku zmienne typu own, deklarowane lokalnie, których "czas życia" jest tak długi, jak długo trwa wykonywanie całego programu, w którym te zmienne występują. Jednakże nie bez powodu wiele kompilatorów ALGOL-u skonstruowano tak, że zmiennych typu own nie "tolerują". "Tolerowanie" takich zmiennych wymaga bowiem traktowania deklaracji, a przynajmniej deklaracji typu own, statycznie. Deklarowanie to nie może odbywać się wtedy, gdy następuje wejście do odpowiedniego bloku, czy wywołanie procedury lub funkcji. Musi ono odbywać się w fazie tłumaczenia programu, a najpóźniej w momencie jego zainicjowania. To zaś narzuca dość kłopotliwe ograniczenia na gospodarkę pamięcią; nie może ona być wtedy w pełni dynamiczna.

Poza tym samo wykorzystywanie zmiennych typu own nastręcza pewne trudności, ponieważ przy każdym wejściu do bloku, w którym one są zadeklarowane, musi następować sprawdzenie, czy mają one wartości, czy też jest to wejście pierwsze, przy którym należy im nadać wartości początkowe.

Powyższe wywody mogą także służyć jako odpowiedź na pytanie, dlaczego w nowszych językach programowania nie mamy zmiennych typu own, a więc nie możemy również rozwiązać problemu zmiennych, które "żyją" dłużej niż procedury nimi operujące, aczkolwiek niedostępne są poza tymi procedurami. Propozycje takiego rozwiązania podał Hoare [4]. Proponuje on mianowicie, by takie zmienne i procedury na nich operujące ująć w pewną całość i potraktować jako nową konstrukcję językową, ktorą nazywa monitorem. Forma ta miałaby postać następującej deklaracji:

```
nazwa monitora: monitor
    begin ... deklaracje zmiennych lokalnych ...;
        procedure nazwa procedury  ... ;
           begin ... ciało procedury ... end;
        ...deklaracje innych procedur...;
        ...nadanie wartości początkowych zmiennym lokalnym ...
        end;
```

Wywołanie którejś z procedur danego monitora odbywałoby się za pomocą następującego zdania:

```
nazwa monitora.nazwa procedury  (... parametry aktualne)
```

Zmienne lokalne w monitorze mają wiele wspólnego ze zmiennymi typu own, gdyż również "żyją" dłużej niż trwa wykonanie, najdłuższej nawet, procedury tego monitora. "Żyją" tak długo, jak długo jest ważna deklaracja całego monitora. Deklarację tę należy jednak rozumieć inaczej niż deklarację procedury czy funkcji. Zadeklarowanie monitora powinno bowiem być w y k o n a n i e m jego programu, tzn.

- zadeklarowaniem zmiennych lokalnych
- zadeklarowaniem procedur
- nadaniem wartości początkowych zmiennym lokalnym.

Mamy zatem próbę rozwiązania problemu zmiennych, które mają być lokalne, jeśli chodzi o dostęp do nich, a jednocześnie globalne, jeśli chodzi o "czas ich życia". Nawiasem mówiąc, z czysto językowego punktu widzenia trudno byłoby uzasadnić, dlaczego procedury wchodzące w skład monitora są dostępne także

poza nim samym, natomiast zmienne w nim zadeklarowane są niedostępne. Innymi słowy dlaczego poprawne jest wywołanie:

nazwa monitora.nazwa procedury (... parametry aktualne...)

natomiast będzie błędem użycie zmiennej:

nazwa monitora.nazwa zmiennej

3. PROCESY RÓWNOLEGŁE

W żadnym z trzech wymienionych języków programowania - ALGOL, PASCAL i PEARL - nie mamy możności wyrażenia tego, że wykonywanie pewnych zdań programu powinno być równoległe. Nie możemy wyrazić nawet tego, że nie musi ono następować w takim porządku, w jakim owe zdania są napisane. Jednakże w literaturze można znaleźć wiele propozycji uzupełnienia tych języków o specjalny nawias, wskazujący możliwość równoległego wykonywania zdań lub wyrażeń nim objętych.

A więc np. Dijkstra [2] zaproponował:

<u>parbegin</u> S_1; S_2;...;S_n <u>parend</u>,

Brinch-Hansen [1] podał propozycję różniącą się od powyższej tylko formalnie

<u>cobegin</u> S_1; S_2;...;S_n <u>coend</u>.

Istotne różnice pomiędzy obiema propozycjami dotyczą zapisu ewentualnej synchronizacji wykonywania zdań S_1, S_2,... S_n. Dijkstra wprowadził w tym celu nowy typ zmiennych - <u>semafor</u> - i dwie operacje działające na takich zmiennych - P i V. Natomiast Brinch-Hansen zaproponował nową postać zdania:

<u>region</u> v <u>do</u> ... część programu operująca na zmiennej v ... <u>end</u>

określająca dla danego procesu jego przedział krytyczny ze względu na zmienną v.

Nie będziemy tutaj szerzej omawiać obu propozycji, gdyż zostały one szczegółowo wyjaśnione w cytowanych pracach. Zwróć-

my natomiast uwagę na niektóre problemy wynikające z rozważań o znaczeniu zdania postaci:

<u>cobegin</u> S_1; S_2;...; S_n <u>coend</u>

W szczególnym przypadku może to być np. zdanie:

<u>cobegin</u> S(a); S(b) <u>coend</u>,

gdzie S jest uprzednio zadeklarowaną procedurą z jednym parametrem formalnym. Program stanowiący ciało tej procedury może być wykonywany jednocześnie przez dwa procesory, przy czym oba wykonania (procesy) odnoszą się do różnych parametrów aktualnych. Zatem jeden parametr formalny ma być związany na raz z dwoma aktualnymi.

Najprostszym rozwiązaniem tego problemu jest reguła kopiowania ciała procedury w miejsce jej wywołania wraz z zamianą parametrów formalnych na aktualne. Zauważmy, że deklaracje zmiennych, jakie mogą wystąpić w ciele tej procedury muszą być również kopiowane tak, by zadeklarowane zostały dwa różne komplety zmiennych, mimo iż z tymi samymi nazwami.

Rozważmy teraz inne rozwiązanie, w którym ciało procedury nie będzie kopiowane w miejsce jej wywołania, natomiast program zawarty w jej ciele będzie rzeczywiście wykonywany przez dwa procesory na raz. Jednakże w samym programie nie może być informacji o dwóch na raz parametrach, ani też o dwóch na raz kompletach zmiennych w nim zadeklarowanych. Jeżeli nawet mogłoby tak być, to powstałby problem, jak dany procesor miałby być związany z jednym parametrem i jednym kompletem zmiennych przez cały czas wykonywania tego programu. Widać stąd, że każdy procesor musi zawierać w sobie niedostępną dla innych informację o tym, z którym parametrem aktualnym i którym kompletem zmiennych ma do czynienia wykonując program takiej procedury.

Zatem przyjmując to drugie rozwiązanie dochodzimy do czegoś, co na poziomie języka maszynowego określa się często jako rejestr bazowy procesora. Zauważmy, że taki rejestr jest konieczny nawet wówczas, gdy procesy są tylko quasirównoległe, realizowane przez jeden procesor. Mają one bowiem być między sobą

synchronizowane, a więc jeden wstrzymywany a drugi wznawiany, co oznacza, że również i w tym przypadku będą "istnieć obok siebie" oba parametry aktualne i oba komplety zmiennych.

W przypadku wielu procesorów, jak już wykazaliśmy, rejestr ów musi być częścią składową każdego z nich, natomiast w przypadku procesów quasirównoległych wystarczy, by był to pointer wskazujący, na którym parametrze i którym komplecie zmiennych procesor ma działać. Wartość tego pointera ulega odpowiedniej zmianie, gdy następuje przełączenie procesora z jednego procesu na drugi.

Mówiąc o procesach wykonywanych równolegle lub quasirównolegle powinniśmy zdawać sobie sprawę, że już z góry zakładamy istnienie pewnego systemu operacyjnego, którego zadaniem jest dokonywać przydziału procesorów poszczególnym procesom (czyli przydziału procesorów do wykonywania zdań S_1, S_2,..., S_n). A w przypadku, gdy procesorów będzie mniej niż procesów, wtedy ów system operacyjny będzie procesory "rozmnażać" (czyli dokonywać tzw. podziału czasu). Zatem przyjęcie powyższych propozycji zapisu zdań współbieżnych sprawiłoby, że byłby to język nieco zbyt wysokiego poziomu, przynajmniej jeśli chodzi o pisanie programów przydziału i "rozmnażania" procesorów.

4. PRZYDZIAŁ PROCESORA

Oczywiście przydział procesora (czyli zainicjowanie lub wznowienie procesu), tj. skierowanie go do wykonywania jakiegoś zdania programu, nie odbywa się poprzez zdanie go to. Mówiąc o przydziale procesora mamy na myśli to, że procesor "dochodzi" do tego zdania w inny sposób niż to wynika z dynamicznego porządku wykonywania zdań składających się na dany program.

Rozważmy zdanie z poprzedniego paragrafu:

<u>cobegin</u> S_1; S_2;...; S_n <u>coend</u>

i załóżmy, że zanim doszło do wykonywania tego zdania, realizowany był tylko jeden proces, a więc potrzebny był tylko je-

den procesor. W procesie tym być może doszło do deklaracji jakiejś zmiennej. Zatem owa zmienna ma być wspólna dla równoległych procesów, stanowiących wykonanie zdań S_1, S_2,..., S_n. Oznacza to, że ma ona być dostępna nie tylko dla procesora, który ją zadeklarował, ale również dla tych procesorów, które zostaną przydzielone do wykonywania zdań S_1, S_2,..., S_n. Wobec tego przydział procesora powinien być m.in. "powiadomieniem" go o istnieniu tej zmiennej. Innymi słowy, ma on "wiedzieć" wszystko to samo, co "wie" ów pierwszy procesor. Zatem przydział procesora miałby być równoznaczny z jego "utożsamieniem się" z tym procesorem, który do tej pory realizował program.

Jednakże całkowite "utożsamienie się" oznaczałoby dalej realizowanie dokładnie tego samego procesu (wykonywanie tego samego programu). A przecież chodzi o to, by różne procesory były przydzielone do różnych procesów. Z tego wniosek, że owo "utożsamienie się" nie może być całkowite.

Powstaje teraz problem określenia, jak to "utożsamienie się" rozumieć i których zmiennych oraz parametrów ma dotyczyć. Rzecz powinna być jednak rozważona w powiązaniu z problemem, jak rozumieć odbieranie procesora procesom (np. wtedy, gdy proces się zakończył) oraz jaki program wykonuje procesor, kiedy przestał już realizować jeden proces a jeszcze nie zaczął innego. Jeżeli założymy, że jest to program oczekiwania na informacje od innego procesora (np. informacje o tym, że jakiś proces może być zainicjowany lub wznowiony), to cały problem musimy znowu rozwiązywać od początku. Przekazywanie informacji oznacza bowiem istnienie wspólnych zmiennych, a przecież problem wyniknął z tego, jak procesor ma się "dowiedzieć" o istnieniu wspólnej zmiennej.

Zauważmy, że na niższym poziomie języków programowania, gdzie operuje się pojęciami pamięć maszyny i adresy poszczególnych słów, czy bloków, problem nie istnieje. Cała pamięć może być traktowana jako wspólna dla wszystkich procesorów. Jeżeli zatem deklaracje będą traktowane statycznie, tzn. będą to jedynie informacje dla kompilatora, który już w fazie trans-

lacji rozplanuje położenie wszystkich zmiennych (i programów), to nie będzie problemów operacji na pamięci związanych z deklarowaniem zmiennych, a więc i problemu "utożsamienia się" jednego procesora z drugim. Przydzielenie procesora do procesu powinno wówczas niewiele różnić się od go to. Byłoby to jednak z pogwałceniem kardynalnej zasady mówiącej, że za pomocą go to nie powinno się wchodzić do wnętrza bloku, procedury czy funkcji (nie mówiąc już o tym, że zdaniem wielu autorów, stosowanie go to powinno być w ogóle zabronione). Poza tym, jak wspomnieliśmy w paragrafie 2, deklaracje traktowane statycznie sprawiają pewne kłopoty, a jeśli dotyczą każdego rodzaju zmiennych, to praktycznie uniemożliwiają rekursję.

Problem przydziału i odbierania procesora ma jeszcze jeden, być może nawet głębszy aspekt. Otóż z punktu widzenia systemu operacyjnego problem ten to problem sterowania procesami, to znaczy wykonania programów użytkowników. Natomiast z punktu widzenia użytkowników system operacyjny świadczy usługi, to znaczy ich programy niejako sterują systemem operacyjnym. Dlatego też przyjęło się mówić o wywoływaniu procedur systemu operacyjnego. A przecież, z pierwszego punktu widzenia, można równie dobrze mówić o wywoływaniu programów użytkowników. Wydaje się, że każdy, kto miał do czynienia z programowaniem systemów operacyjnych, uświadamiał sobie ten dylemat. Ponieważ jednak pisał programy w języku maszynowym lub do niego zbliżonym, posługiwał się pojęciami takimi jak adresy i podprogramy. A to pozwalało na odwracanie wspomnianej hierarchii w miarę potrzeby lub też na sprowadzenie wszystkiego do jednego poziomu rozumowania. Natomiast w stosowanych dotychczas językach programowania wyższego poziomu trzeba wyraźnie określić hierarchię sterowania; nawet jeżeli procedura a będzie wywoływać procedurę b, a ta z kolei procedurę a, nie oznacza to odwrócenia hierarchii, tylko rekursję.

5. ZAKOŃCZENIE

Brak w literaturze pozycji, w których można byłoby znaleźć przykład zaprogramowania całego systemu operacyjnego w jakimś jednym języku wyższego poziomu, zdaje się potwierdzać mniemanie autora o nieprzydatności takich języków w ich obecnej postaci do tego celu.

Nie oznacza to jednak, by nie podejmowano prób strukturalnego programowania systemów operacyjnych. Sposób postępowania przy takim programowaniu bardzo jasno przedstawił Dijkstra [3] w pracy tylokrotnie już na tym sympozjum cytowanej. W wielkim skrócie metodę tę można przedstawić jako konstruowanie kolejnych, coraz wyższych poziomów oprogramowania wychodząc od języka maszyny. Innymi słowy jest to definiowanie kolejnych języków programowania danej maszyny korzystając za każdym razem z języka poziomu niższego. Zatem system operacyjny powstaje jako pewien hierarchiczny zbiór programów, napisanych jednak w różnych językach, odpowiednio do poziomu, na którym program znajduje się w całej hierarchii.

Literatura

[1] BRINCH-HANSEN P.: Structured Multiprogramming, CACM, 1972, t. 15, nr 7.

[2] DIJKSTRA E.W.: Cooperating Sequential Processes, in: F. Gennys (ed) Programming Languages, Acad. Press. 1968.

[3] DIJKSTRA E.W.: The Structure of the THE Multiprogramming System, CACM 1968, t. 11, nr 5.

[4] HOARE C.A.R.: Monitors: an Operating System Structuring Concept, 1973, informacja prywatna.

[5] NAUR P. i inni: Report on the Algorithmic Language ALGOL 60, CACM, 1960, t. 3, s. 299-314.

[6] SNOWDON R.: PEARL - Program Elaboration and Refinement Language, University of Newcastle u/Tyne, T.R., 1971, nr 28.

[7] WIRTH N.: The Programming Language PASCAL, Acta Informatica, 1971, t. 1, nr 1.

ПРОБЛЕМЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ ОПЕРАЦИОННЫХ СИСТЕМ

Резюме

Структурным программированием принято определять такой метод программирования, который позволяет получать программы в виде четкой документации. Четкость документации достигается путем хорошего определения структуры программы. На каждом уровне этой структуры нас интересует только то, что данная программа (процедура, подпрограмма) выполняет, а не каким образом. Обсуждение того, каким образом, является переходом к следующему уровню структуры, на котором снова появляются программы (процедуры, подпрограммы). Ответ на вопрос: что ими выполняется, одновременно является ответом на вопрос: каким образом выполняется программа предыдущего уровня. Элементарные операции данной машины или языка программирования должны являться поооледним уровнем структуры.

Значительную помощь в таком составлении программ оказывают процедурные языки программирования, напр. ALGOL, PASCAL а в последнее время общий интерес вызывает язык PEARL , предлагаемый в виде языка структурного программирования.

С другой стороны известно, что до сих пор операционные системы, а по крайней мере их базовые части, программируются на машинных языках или на языке ассемблера, без использования структурного метода. Документация системы всегда является серьезной проблемой и нуждается, кроме текстов самых программ, еще в их описании на неформальном языке.

В настоящей разработке автор пытается ответить на вопрос, почему процедурные языки в их настоящей форме, а даже в форме специально подобранной для представления частей операционной системы, непригодны для их программирования.

PROBLEMS OF STRUCTURED PROGRAMMING OF OPERATING SYSTEMS

Summary

Structured programming is meant as a programming method that renders, among others, a program text to be its own, readable documentation. It becomes possible only when the program structure is stated clearly. At every level of the structure we are interested only in what a given program (procedure or subroutine) does, not in how it does. A "how" consideration means going to the next level of the structure, where again programs (procedures or subroutines) are to be dealt with. The answer to the question what they do is the answer to the question how they do at the former level. Elementary operations of a given machine or in a programming language become the last level of the structure.

Procedural programming languages - e.g. ALGOL, PASCAL - provide a programmer with a substantial help in such method of programming. Recently, a language PEARL has been discussed as a tool specially developed for structured programming. On the other hand operating systems or at least their basic parts are programmed in machine languages or assemblers, in the way that is far from the structured one. Operating system documentation is always a big problem which is being solved in the form of descriptions written, unfortunately, in unformalized languages.

The content of the present paper is an attempt to answer the question why procedural languages in their today forms, or even in forms especially proposed for describing parts of operating system, are not used for their programming.

© Instytut Maszyn Matematycznych
02-076 Warszawa, ul. Krzywickiego 34

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

ALGORYTMY, Zeszyt Specjalny, 1974, 173-180

681.322.06:
061.3/438/"1973"

ROZSZERZENIE MOŻLIWOŚCI FUNKCJONALNYCH PROGRAMU EGZEKUTOR DLA EMC ODRA 1304

Janusz PIELA
Wojciech SKURZAK

Instytut Automatyzacji Systemów Zarządzania
Wojskowej Akademii Technicznej

Pracę złożono 10.I.1974

Modyfikacje Egzekutora mają na celu rozszerzenie jego własności funkcjonalnych i mogą być prowadzone w dwóch podstawowych kierunkach:

- dołączanie oprogramowania niestandardowych urządzeń zewnętrznych,
- wprowadzenie ułatwień programowych, umożliwiających konstruowanie fragmentów systemu operacyjnego, działających na poziomie programu użytkowego.

Zmiany wprowadzone do Egzekutora wykorzystano w praktycznie opracowanych systemach.

S p i s   t r e ś c i

1. WSTĘP

Rozwijające się różne dziedziny zastosowań maszyn cyfrowych stawiają coraz większe wymagania zarówno w zakresie sprzętu jak i oprogramowania. Często zachodzi konieczność budowy wąsko wyspecjalizowanych systemów, których elementami powinny być specyficzne dla nich urządzenia.

Firmowe oprogramowanie dostarczane przez producentów maszyn cyfrowych ma najczęściej charakter uniwersalny i nie spełnia zadań stawianych przed systemami wyspecjalizowanymi. Powstaje więc problem budowy systemów operacyjnych dla specjalnych zastosowań maszyn cyfrowych.

Opracowanie od podstaw takich systemów jest bardzo czasochłonne oraz wymaga dużego nakładu sił i środków. Cechą charakterystyczną współczesnego oprogramowania powinna być możliwość jego modyfikacji w celu dostosowania do nowych zastosowań. Pozwala to na wprowadzenie nowych elementów oprogramowania przy zachowaniu wszystkich możliwości funkcjonalnych oprogramowania firmowego.

Cechę taką posiada oprogramowanie maszyn cyfrowych systemu ODRA 1300. Producent oprogramowania zapewnia, że istnieje możliwość modyfikacji programu nadzorczo-sterującego, jakim jest program Egzekutor, jak również możliwość rozbudowy bibliotek podprogramów obsługiwanych przez kompilatory poszczególnych języków programowania. Modyfikacji programu Egzekutor można dokonywać przy codziennej eksploatacji systemu. Jednak do wykonania nawet najprostszych modyfikacji Egzekutora konieczna jest znajomość zasad jego pracy i budowy.

Analizę możliwości modyfikacji Egzekutora przeprowadzono w związku z koniecznością dołączenia oprogramowania niestandardowych urządzeń zewnętrznych, np. grafoskopu, urządzeń transmisji danych. Głębsza analiza wykazała, że przez modyfikację można również zmienić własności funkcjonalne Egzekutora, zachowując całe dotychczasowe oprogramowanie, tzn. kompilatory i standardowe programy użytkowe.

Nowe funkcje dołączone do standardowego Egzekutora wykorzystywane są najczęściej za pośrednictwem nowych ekstrakodów. Ekstrakody te można bezpośrednio stosować w językach symbolicznych np. NSBL, PLAN. Można ich również używać w programach napisanych w językach wysokiego poziomu (np. ALGOL, COBOL, FORTRAN) stosując wstawki w języku PLAN.

2. TABLICE PROGRAMU EGZEKUTOR

Egzekutor w czasie pracy maszyny znajduje się stale w ohronionym obszarze pamięci operacyjnej. Głównymi funkcjami Egzekutora są:

1. nadzorowanie wykonywania programów,
2. nadzorowanie pracy urządzeń zewnętrznych,
3. realizacja ekstrakodów,
4. wykonywanie akcji specjalnych (np. ładowanie programu),
5. komunikacja z operatorem i wykonywanie jego poleceń.

W programie Egzekutor przechowywane są informacje dotyczące wszystkich programów znajdujących się w pamięci operacyjnej maszyny. Informacje o programie można podzielić na dwie grupy:

- informacje identyfikujące program i określające przydzielone programowi urządzenia zewnętrzne oraz wielkość obszaru pamięci,
- informacje określające aktualny stan programu.

Informacje te przechowywane są w tablicach opisu programów w obszarze Egzekutora. Oprócz opisu programów znajdujących się w pamięci operacyjnej maszyny, konieczny jest również opis urządzeń zewnętrznych dołączonych do maszyny. Informacje służące do opisu urządzeń zewnętrznych możemy podzielić na dwie grupy:

- stałe, określające własności danego urządzenia zewnętrznego i sposoby odwołania się do niego,

- zmienne, określające stan urządzenia zewnętrznego w każdej chwili czasu i przynależność do określonego programu.

Informacje te znajdują się w kilku różnych tablicach opisu urządzeń w obszarze Egzekutora.

Tablice opisu programu i opisu urządzeń zewnętrznych wykorzystywane są przez podprogramy Egzekutora. W celu skrócenia czasu dostępu do żądanych informacji są one rozmieszczone w kilku różnych tablicach powiązanych logicznie ze sobą.

3. MODYFIKACJE PROGRAMU EGZEKUTOR

Modyfikacje programu Egzekutor mają na celu rozszerzenie jego własności funkcjonalnych. Mogą one dotyczyć:

- oprogramowania dołączonych do maszyny niestandardowych urządzeń zewnętrznych,
- wprowadzania ułatwień programowych umożliwiających konstruowanie fragmentów systemu operacyjnego na poziomie programów użytkowych.

Uzupełnienia programu Egzekutor mogą być wykonane:

- jako niezależne pakiety wymagające jedynie określenia punktów wejścia i wyjścia w dotychczasowym Egzekutorze,
- jako niezależne pakiety wykorzystujące dotychczasowe właściwości i podprogramy Egzekutora.

3.1. Oprogramowanie niestandardowych urządzeń zewnętrznych

Technicznemu dołączeniu niestandardowych urządzeń zewnętrznych do maszyny systemu ODRA 1300 musi towarzyszyć oprogramowanie ich na poziomie Egzekutora. Ważne jest aby sposób współpracy z tymi urządzeniami nie odbiegał od sposobu współpracy z urządzeniami standardowymi. Polega to m.in. na stosowaniu typowych ekstrakodów. W niektórych przypadkach celowe byłoby

stosowanie dodatkowych ekstrakodów. W ogólnym przypadku jednak dla zapewnienia możliwości wykorzystywania niestandardowych urządzeń w programach pisanych za pomocą różnych języków programowania należy zachować istniejące standardy. Opracowanie pakietu oprogramowania nowego urządzenia zewnętrznego nie jest zbyt trudne. Pakiet taki składa się z dwóch podstawowych elementów:

- opisu urządzenia,
- podprogramów współpracy z urządzeniem.

Opis urządzenia w pakiecie stanowią informacje będące pozycjami poszczególnych tablic opisu urządzeń zewnętrznych w Egzekutorze. Przy opracowaniu oprogramowania nowego urządzenia zewnętrznego należy włączyć do istniejących tablic Egzekutora pozycje charakteryzujące dołączane urządzenie. Informacje umieszczone w tablicach opisu urządzenia wykorzystywane są przez podprogramy gospodarki urządzeniami zewnętrznymi, podprogramy komunikacji z operatorem, jak również podprogramy inicjowania i kończenia transmisji. Oprogramowanie każdego urządzenia powinno uwzględnić jego specyficzne cechy. Z tego powodu nie można ściśle określić standardu oprogramowania wszystkich urządzeń zewnętrznych.

W przypadku opracowania nowych ekstrakodów współpracy z dołączonym urządzeniem można wykorzystać kody ekstrakodów niewykorzystane w konkretnej wersji Egzekutora. Oprogramowanie dołączonego urządzenia stanowi wówczas niezależny podprogram. Punktami wspólnymi z Egzekutorem są tylko:

- punkt wejścia do podprogramu obsługi urządzenia z podprogramu Egzekutora dekodującego poszczególne ekstrakody,
- punkt wyjścia z dołączonego podprogramu do podprogramu Egzekutora realizującego powrót do programu użytkowego.

Praktycznie przeprowadzono dwie modyfikacje. Pierwsza z nich umożliwia wykorzystanie systemowej konsoli operatora do współpracy z programem użytkowym. Celem tej modyfikacji było stworzenie możliwości prowadzenia eksperymentów w zakresie organi-

zacji pracy konwersacyjnej przy wykorzystaniu dostępnego zestawu maszyny. Zrealizowany został dodatkowy ekstrakod, który pozwala na wprowadzenie i wyprowadzenie informacji ze wskazanego obszaru pamięci operacyjnej programu użytkowego. Druga modyfikacja umożliwiła dołączenie do maszyny cyfrowej ODRA 1304 urządzenia transmisji danych UTD-211. W obecnej chwili opracowywane jest oprogramowanie umożliwiające współpracę maszyny cyfrowej ODRA 1304 poprzez urządzenie UTD-211 z minikomputerem MOMIK-8b.

3.2. Wprowadzanie ułatwień programowych umożliwiających konstruowanie fragmentów systemu operacyjnego na poziomie programu użytkowego

W tradycyjnej pracy z maszyną cyfrową ODRA 1304 wykonywanie programów nadzorował operator maszyny, nadając odpowiednie komunikaty do Egzekutora oraz śledząc wydruki na konsoli. Tak zorganizowana praca znacznie wydłuża proces przetwarzania oraz wymaga od operatora znacznego wysiłku umysłowego. Ponieważ w maszynie ODRA 1304 można przetwarzać równocześnie cztery programy, funkcje operatora mógłby przejąć jeden program i sterować pracą pozostałych trzech. Takie sterowanie pracą programów wymaga, żeby wybrany program nazywany dalej sterującym mógł nadawać komunikaty operatorskie do pozostałych trzech programów nazywanych dalej sterowanymi oraz, żeby informacje o zdarzeniach jakie zajdą w programach sterowanych były przekazywane do tego jednego wybranego programu. Aby to zapewnić opracowano nowe ekstrakody o następujących znaczeniach:

- wyróżnij program - ekstrakod ten powoduje zaznaczenie w tablicach opisu programu w Egzekutorze, że program jest wyróżniony, co oznacza, że będą do niego przekazywane informacje o rozpoznanych przez Egzekutor zdarzeniach w sterowanych programach,
- wykonaj komunikat z pola programu - ekstrakod ten powoduje wykonanie przez Egzekutor komunikatu operatorskiego, który podany jest w obszarze programu sterującego. Dopuszczalne są

wszystkie komunikaty operatorskie; komunikaty te mogą dotyczyć ładowania programów, inicjowania ich pracy, zawieszania itp.

- zawieś program sterujący - ekstrakod ten pozwala na zawieszanie programu sterującego i przejście do wykonania innych programów. Program sterujący pozostaje zawieszony aż do wystąpienia w którymś z programów sterowanych jednego z określonych zdarzeń,

- usuń wyróżnienie programu - ekstrakod ten pozwala na usunięcie znacznika z tabel Egzekutora oznaczającego, że program jest programem sterującym pracą innych programów. Po wykonaniu tego ekstrakodu nadzór nad wszystkimi programami przejmuje operator.

Za pomocą powyższych ekstrakodów opracowano już dwa programy sterujące. Pierwszym z nich jest program sterujący pracą systemu przetwarzania danych. Program ten zajmuje zaledwie około 300 słów pamięci operacyjnej. Drugim jest System Automatycznego Uruchamiania Programów. Jest on przystosowany do kompilatorów języka PLAN.

Opracowane ekstrakody umożliwiają pisanie programów sterujących na poziomie programu użytkowego w dowolnym dostępnym w systemie ODRA 1300 języku programowania np. PLAN, ALGOL, FORTRAN itp. Pozwala to na praktyczne sprawdzenie koncepcji w krótkim czasie.

4. WNIOSKI

Drogą modyfikacji można znacznie rozszerzyć możliwości programu Egzekutor. Nie jest to zbyt trudne, jeżeli znana jest budowa i zasady pracy programu Egzekutor. Egzekutor z omówionymi zmianami eksploatowany jest już przez pół roku i nie stwierdzono zakłóceń w pracy programów, które nie korzystają z wprowadzonych udogodnień.

РАСШИРЕНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ EXECUTIVE ДЛЯ ЭВМ ODRA-1304

Резюме

Модификация программы EXECUTIVE имеет целью расширить ее функциональные свойства и может проводиться по двум основным направлениям:

I. Подключения матобеспечения нестандартных внешних устройств ;
2. Введения программных средств, позволяющих разработать элементы операционной системы, функционирующие на уровне прикладной программы.

Изменения, введенное в программу EXECUTIVE, нашли практическое применение в разрабатываемых системах.

INCREASE OF FUNCTIONAL FACILITIES OF THE ODRA 1304 EXECUTIVE

Summary

The modifications of the Executive are introduced to give additional functional facilities for its users. The modifications can be added in two following ways:

1. adding the software for non-standard peripherals,
2. introducing the elements of software, enabling one to write control programs at user s program level.

These modifications have been tested practically.

ALGORYTMY, Zeszyt Specjalny, 1974, 181-192

681.322.06:
061.3/438/"1973"

ORGANIZACJA ZINTEGROWANYCH MANIPULATORÓW

Zbigniew KOSOWSKI
Krzysztof BYTNEROWICZ

Zakład Doświadczalny Oprogramowania IMM

Pracę złożono 10.I.1974

Istotnym problemem przy tworzeniu biblioteki systemowych programów manipulacyjnych jest ich liczba i różnorodność sposobów korzystania przy jednoczesnym często podobnym działaniu. W opracowaniu przedstawiono ideę zintegrowanych manipulatorów pozwalającą zredukować liczbę programów manipulacyjnych do kilku oraz ujednolicić sposób korzystania z nich. Omówiona została koncepcja budowy programu głównego spełniającego szerokie funkcje przez wywoływanie modułów realizujących wyspecjalizowane zadania, przy czym program główny i moduły realizujące mogą być napisane w różnych językach programowania. Podstawą opracowania stały się praktyczne doświadczenia wynikłe z pracy przy serwisie oprogramowania maszyny ZAM 41 oraz w trakcie tworzenia biblioteki oprogramowania w systemie OS/360.

S p i s t r e ś c i

1. MANIPULATORY I ICH MIEJSCE W SYSTEMIE OPROGRAMOWANIA EMC

W niniejszym opracowaniu przyjęto założenie, że maszyna cyfrowa jest wyposażona w oprogramowanie podstawowe. Rozumiemy przez to, że istnieją już programy supervisora, programy realizacji metod dostępu (data management) oraz translatory pewnych języków. Nadmienić należy, iż w tych dopiero warunkach integracja manipulatorów w pełni daje efekty.

Programem manipulacyjnym lub manipulatorem (utility) nazwiemy taki program, który spełnia powtarzalne pomocnicze funkcje związane z eksploatacją pewnego konkretnego systemu oprogramowania na konkretnej maszynie cyfrowej. Są to więc programy związane z obsługą bibliotek, konserwacją nośników i voluminów, edycją wszelkich tekstów w formach zależnych od tych tekstów np. edycja programów zapisanych w konkretnym języku z uwzględnieniem struktury tego języka , programy konwersji danych pomiędzy nośnikami, programy aktualizacji tekstów itp.

Na podstawie tej charakterystyki można więc stwierdzić, iż są to w większości programy, w których dużą część czasu pracy zajmują operacje wejścia-wyjścia.

Ze względu na charakter zastosowań są to programy, z którymi ma do czynienia prawie każdy użytkownik danego systemu oprogramowania, tak więc bardzo istotne są wszelkie korzyści, które można uzyskać przez odpowiednie opracowanie tych programów.

2. MANIPULATORY W MASZYNACH WCZESNYCH GENERACJI

W związku z ubogim wyposażeniem EMC O i I generacji w urządzenia wejścia-wyjścia i ich stosunkowo małą wydajnością można chyba bez ryzyka stwierdzić, że podstawowym edytorem tych maszyn był w najlepszym przypadku po prostu dalekopis lub opisywacz kart. Pozostałe funkcje były raczej tak nikłe, że można je pominąć. Maszyny II i III generacji w dużym stopniu

zmieniły sytuację w tej dziedzinie. Dzięki wprowadzeniu stosunkowo szybkich urządzeń wejścia-wyjścia oraz wzrostowi szybkości EMC zaczęto je również stosować do wykonywania funkcji pomocniczych. Następnym faktem wpływającym na rozwój programów manipulacyjnych była rosnąca w ogromnym tempie liczba informacji przechowywanych na maszynowych nośnikach. Powodowało to konieczność stosowania specjalnych programów umożliwiających wizualną kontrolę zawartości nośników i ich maszynową aktualizację.

Szczególnie w przypadku programów edycyjnych doprowadziło to do takiej sytuacji, że niejednokrotnie każdy typ informacji miał "swego" edytora.

Ciekawym przykładem jest tu system oprogramowania SO141 zrealizowany w IMM na EMC ZAM 41, który zawiera:

- 2 programy aktualizacji tekstów (POPR, SMAO),
- 11 edytorów, w tym 3 obsługujące programy napisane w MSAS w zależności od kombinacji wejścia-wyjścia,
- 4 manipulatory związane z obsługą taśmy bibliotecznej systemu oprogramowania SO141,
- 8 manipulatorów związanych z tekstami zapisanymi w standardzie SMAD,
- 3 manipulatory związane z tekstami zapisanymi w standardzie SMAO,
- około 15 manipulatorów związanych z informacją zapisaną na taśmach magnetycznych (TM) w standardzie IMM (grupa OPUS i inne). Jest to grupa o tyle nietypowa, że zawiera oprócz programów konwersji i sprawdzania danych również programy sortowania, generacji wydawnictw oraz metrykowania, powielania i sprawdzania taśm magnetycznych.

Przyznać tutaj należy, iż różnorodność ta jest w pewnym stopniu również "zasługą" autorów niniejszego opracowania.

Na swe usprawiedliwienie możemy dodać, że jak nam wiadomo, owa różnorodność dotyczy nie tylko EMC ZAM 41. Już w trakcie pracy na ZAM 41 autorzy podjęli udaną chyba próbę opracowania "uniwersalnego edytora" (VWYD), który zawiera w sobie kilka reżimów edycyjnych, a dzięki wykorzystaniu aparatu symbolicznych operacji wejścia-wyjścia może spełniać funkcje konwersji szczególnie w dziedzinie przekodowywania informacji z jednych kodów zewnętrznych na inne. Manipulator ten spełnia następujące funkcje:

- edycja tekstów bez stronicowania,
- edycja tekstów ze stronicowaniem,
- wydruk kart perforowanych ze stronicowaniem i numerowaniem,
- wydruk tekstów ze stronicowaniem i opatrywaniem stron w nagłówki,
- wydruk programów w MSAS, PJEG, z oddzieleniem czołówki JOM oraz numerowaniem wierszy programu,
- proste powielanie taśmy papierowej,
- powielanie taśmy z redagowaniem taśmy papierowej,
- wielokrotny wydruk tekstów.

Jak jednak wykażemy dalej, z naszego punktu widzenia, program ten jest nie tyle zintegrowanym manipulatorem edycyjnym, ile próbą integracji kilku programów edycyjnych w jedną fizyczną całość. Z drugiej strony dodać chyba należy, iż na tym poziomie rozwoju oprogramowania jaki reprezentuje SO141 posiadanie takiego programu jest już dużą wygodą.

W tym samym SO141 autorzy opracowali grupę programów spełniających różne funkcje a związanych ze sobą podmiotem działalności (teksty zapisane w standardzie SMAD) oraz sposobem korzystania. Programy te spełniają funkcje:

- perforowanie wybranych elementów z TM zapisanej w standardzie SMAD,
- scalanie i powielenie taśm SMAD,
- metrykowanie i zakańczanie taśm SMAD,
- sprawdzanie i produkowanie spisu stron taśmy SMAD,
- zapisywanie taśm binarnych w standardzie SMAD.

Reasumując można stwierdzić, że w przypadku programu edycyjnego nastąpiła integracja fizyczna, a w przypadku powyższej grupy programów - integracja logiczna.

3. NOWE MOŻLIWOŚCI III GENERACJI EMC

Dla programisty przejście od II do III generacji odbyło się nie w aspekcie sprzętu, leoz jakościowo innego oprogramowania. Mamy tu na myśli przede wszystkim ogromny postęp w standaryzacji operacji wejścia-wyjścia na poziomie programu użytkowego.

Te dwa wspomniane wyżej czynniki pozwalają, zdaniem autorów, lepiej zorganizować programy manipulacyjne.

Otwiera się tu mianowicie możliwość oddzielenia funkcji organizacyjnych związanych z obsługą wejścia-wyjścia od funkcji logicznych związanych ze strukturą przetwarzanych danych.

Idąc dalej można zbudować ramowy program zajmujący się jedynie obsługą wejścia-wyjścia i zależnie od wysterowania lub informacji zawartej w samym tekście przetwarzanym, wywołujący ze swej biblioteki programy odpowiednie do konkretnego typu tekstu. Nie wyklucza się tu oczywiście wielostopniowego powiązania takich programów. Rozwiązanie takie jest możliwe dopiero w systemach oprogramowania, w których każdy program z natury rzeczy jest jednocześnie przystosowany dzięki aparatowi systemu operacyjnego do pracy jako podprogram innego programu, który przekazał mu sterowanie.

4. CEL INTEGRACJI MANIPULATORÓW

Jak wynika z powyższego funkcje wspólne na przykład dla wszystkich edytorów można przenieść do programu ramowego, a w programach do przetwarzania różnych typów tekstów zostawić tylko funkcje logicznie związane ze strukturą tekstu. Uzyskujemy w ten sposób następujące udogodnienia:

- modularność systemu manipulatorów na poziomie programów:
  a. mając program ramowy można uzyskać dużą oszczędność podczas realizacji grupy programów pozbawionych powtarzalnych funkcji organizacyjnych,
  b. duże oszczędności w liczbie rozkazów,
  c. ujednolicenie sterowania i co za tym idzie ułatwienie obsługi,

- możliwość napisania programu ramowego i programów realizacyjnych w różnych językach np. ramowy w języku ASSEMBLER a realizacyjny w COBOL-u,

- dużo większą łatwość dołączania i uruchamiania programu realizacji współpracującego z uruchomionym i sprawdzonym programem ramowym,

- możliwość wykorzystania programów realizacji przez programy ramowe spełniające różne funkcje,

- duża mobilność systemu manipulatorów uzyskana przez uniezależnienie programów realizacyjnych od operacji wejścia-wyjścia w danym systemie oprogramowania. Zmianie musi ulec tylko program ramowy.

Głównym więc celem będzie tutaj otrzymanie maksymalnej elastyczności funkcjonalnej grupy programów, uproszczenie obsługi oraz minimalizacja liczby rozkazów maszynowych zużytych na grupę czynności (np. na czynności edycyjne).

Warunkiem takiej integracji będzie więc przede wszystkim możliwość wywołania każdego programu przez inny program, jako jego podprogram oraz ustalenie jednolitej ale możliwie uniwersalnej i elastycznej metody przekazywania sterowania i informacji do opracowania pomiędzy programem ramowym i programami realizacji. Następnym warunkiem fizycznej integracji jest takie udoskonalenie oprogramowania podstawowego, że cała konwersja informacji zależnej od typu urządzenia wejścia-wyjścia i użytego nośnika lub kodu zewnętrznego na kod wewnętrzny maszyny odbywa się na poziomie metod dostępu.

Podstawowym natomiast warunkiem integracji logicznej jest opracowanie dość szczegółowej koncepcji całości grupy manipulatorów wchodzących w skład danego systemu oprogramowania przed przystąpieniem do ich realizacji, a nie odwrotnie, jak to często ma miejsce w praktyce.

5. STAN REALIZACJI ZINTEGROWANYCH MANIPULATORÓW NA EMC IBM 360, 370 w ZDO IMM (W SYSTEMIE OS/360)

A. Integracja fizyczna

Obecnie w ZDO IMM realizowane są dwa programy ramowe oraz wiele programów realizacji:

1. programy ramowe
   - program edycyjny
   - program aktualizacji tekstów
2. programy realizacji
   - edycja prosta
   - edycja ze stronicowaniem
   - edycja na dwa wyjścia (np. drukarka i perforator kart)
   - wybieranie określonych ciągów dokumentów ze strumienia wejściowego i przekazywanie na wyjście
   - bezkontekstowa zamiana tekstów
   - edycja programów napisanych w języku ASSEMBLER 360
   - edycja opisów oprogramowania zmagazynowanego w formie źródłowej na maszynowych nośnikach informacji.

Do funkcji programu ramowego o nazwie VWYD należy pełna obsługa zbiorów wejścia-wyjścia i przekazywanie sterowania programom realizacyjnym przed wyprowadzeniem każdego z dokumentów na wyjście oraz w momencie końca danych i wykrycia błędów.

Funkcja obsługi zbiorów wejścia-wyjścia polega między innymi na przeczytaniu informacji zawartych w zbiorach opisanych w kolejnych kartach DD czołówki (JCL) zadania, w którym rea-

lizowany jest program. W przypadku taśm magnetycznych program umożliwia odczytanie informacji zawartych w wielu plikach (file) voluminu opisanego za pomocą jednej karty DD, co w przypadku metod dostępu dla taśm magnetycznych w OS/360 jest dość istotne. Sposób współpracy (interface) programu ramowego z programami realizacji, oprócz przekazania dokumentu do przetworzenia, przewiduje również:

- przekazanie informacji o strukturze i organizacji zbioru,
- przekazanie informacji zwrotnej o dokumencie opracowanym,
- przekazanie dyspozycji programowi ramowemu co do czynności jakie ma wykonać:
  a. wypisać opracowany dokument i podać następny,
  b. wypisać opracowany dokument i nie podając następnego oddać sterowanie,
  c. "przewinąć" n kolejnych dokumentów,
  d. zakończyć pracę awaryjnie z powodu niezgodności informacji wejściowej z zadaniami programu realizacyjnego,
  e. zakończyć pracę normalnie,
  f. przerwać współpracę z danym programem realizacyjnym i ewentualnie podjąć z następnym,
- przekazanie programowi realizacji informacji o tym, że struktura dokumentu niezgodna jest z wymaganiami odnośnie budowy zbioru wyjściowego. Program realizacji może poprawić błąd lub zakończyć pracę ze źle zdefiniowanym zbiorem wejściowym i przejść do następnego zbioru wejściowego lub zakończyć pracę, gdy źle zdefiniowano zbiór wyjściowy.

Drugim realizowanym docelowo programem ramowym jest VPOPR będący logiczną kontynuacją programu POPR i standardu SMAD z ZAM 41. Ten program aktualizacji tekstów zapisanych na nośnikach sekwencyjnych z założenia nie jest pomyślany jako interpretator zawartości aktualizowanych tekstów. Przewidywana w nim możliwość współpracy z tymi samymi programami realizacji co i VWYD w dużym stopniu rozszerza jego możliwości. Oprócz realizowanych w nim czynności aktualizacji struktury fizycznej ciągu dokumentów czy znaków w ramach dokumentu (record),

możliwość wywołania programu realizacji związanego z logiczną strukturą aktualizowanego tekstu, rozszerza zakres jego możliwości o czynności interpretacyjne. Ponadto nie obciąża programowo programu ramowego, który realizuje tylko aktualizację fizyczną.

Pracę taką można sobie na przykład wyobrazić następująco: program VPOPR przeprowadza fizyczną aktualizację tekstów a wywołany przez niego program wymiany bezkontekstowej ciągów znaków wymienia w aktualizowanych dokumentach pewne ciągi znaków na inne, przeprowadzając np. podstawianie konkretnych wartości pod symbolicznie oznaczone parametry formalne itp.

Przewidujemy, że po oddaniu do eksploatacji programu VWYD i programów realizacyjnych, biblioteka tych ostatnich szybko zwiększy się o programy redagujące wydruki dla ogólnie stosowanych języków programowania jak ALGOL, FORTRAN, PL/I i inne oraz o inne czynności związane z interpretacją logiczną zawartości dokumentów.

B. Integracja logiczna

Oprócz programów VPOPR i VWYD przetwarzających teksty, opracowywane są w ZDO IMM programy do obsługi maszynowych nośników. Są to programy:

VWTM - program obsługi taśm magnetycznych, którego funkcje polegają na:

- sprawdzaniu zapisu na TM zarówno pod względem poprawności fizycznej jak i zgodności zapisu ze standardem,
- sporządzaniu wypisów zawartości TM poczynając od spisu plików (file), a kończąc na wypisie zawartości zadanych bloków lub dokumentów ,
- metrykowaniu taśm magnetycznych,
- powielaniu taśm magnetycznych,
- testowaniu nośnika.

DIRAC - program obsługi urządzeń o bezpośrednim dostępie, którego funkcje będą polegały na:

- obsłudze zbiorów organizacyjnych VTOC i SYSCTLG umożliwiającej katalogowanie, usuwanie, zmiany nazwy, tworzenie logicznych struktur indeksowych itp.
- produkowaniu wypisów zawartości voluminów dyskowych takich jak: VTOC i SYSCTLG, a wykazy zbiorów PO directory, listy elementów, fizycznej zawartości bloków, ścieżek i cylindrów itp.
- sporządzaniu kopii zbiorów lub voluminów,
- produkowaniu wydruków typu dump z voluminów,
- metrykowaniu i sprawdzaniu dysków.

Integracja logiczna tych programów oraz innych mniej zaawansowanych w opracowaniu polegać ma na tym, iż w odróżnieniu od programów UTILITIES OS/360 będą one miały jednolitą składnię języka sterującego oraz ujednolicone sterowanie za pomocą kart DD języka JCL.

Nadmienić tu warto, że programy manipulacyjne komunikują się z operatorem (programistą) za pomocą wspólnego zestawu ujednoliconych komunikatów o jednolitej składni i z reguły gdy ma to sens pracują w dwóch reżimach:

- standardowym, polegającym na interpretowaniu i wykonaniu kolejnych zdań zestandaryzowanego języka sterującego
- konwersacyjnym, polegającym na kolejnym wykonywaniu poleceń wprowadzanych z konsoli systemowej jako odpowiedzi na standardowe komunikaty emitowane przez manipulatory.

Przełączenie z reżimu standardowego na konwersacyjny odbywa się przez usunięcie karty DD o nazwie "DANE" z ciągu zdań JCL wywołujących dany program manipulacyjny.

6. OSIĄGNIĘTE EFEKTY

Jakkolwiek na tak wczesnym etapie realizacji trudno konkretnie mówić o efektach, to niektóre z nich już w czasie wstępnej eksploatacji i uruchamiania można zauważyć:

- przejęcie przez 4–5 manipulatorów zintegrowanych funkcji kilkunastu manipulatorów firmowych, a w przypadku taśm magnetycznych opracowanie programu, który w bibliotece standardowej OS/360 oprócz funkcji metrykowania nie ma odpowiednika w ogóle,

- wprowadzenie reżimu konwersacyjnego tak ważnego w praktyce operatorskiej, gdzie wiele funkcji realizowanych jest doraźnie bez czasu na przygotowanie programu sterującego,

- ujednolicenie języka sterującego pracą manipulatorów do tego stopnia, iż programy sterujące różnych manipulatorów różnić się powinny jedynie treścią a nie formą. Różnorodność formy jest szczególnie uciążliwa przy korzystaniu z programów grupy UTILITY, gdzie widać dużą pomysłowość programistów poszczególnych programów w komplikowaniu składni języka sterującego,

- wprowadzenie wspólnej grupy komunikatów dla wszystkich manipulatorów w przeciwieństwie do programów UTILITY, gdzie każdy z programów emituje swe własne komunikaty nieraz jednoznaczne z komunikatami innych programów tej samej grupy.

Na zakończenie stwierdzić należy, iż dopiero szeroka praktyka zweryfikuje wyrażone wyżej poglądy i będzie bogatym źródłem doświadczeń w dalszej pracy nad programami manipulacyjnymi, których miejsce w systemie oprogramowania EMC jest przecież tak istotne.

ОРГАНИЗАЦИЯ ИНТЕГРИРОВАННЫХ МАНИПУЛЯЦИОННЫХ ПРОГРАММ

Резюме

Существенная проблема при создании библиотеки системных манипуляционных программ заключается в количестве и разнообразии методов их использования причом их действия неоднократно похожи друг на друга.

В разработке представлена идея интегрированных манипуляторов, позволяющая сократить число манипуляционных программ до нескольких, а также унифицировать метод их использования. Описывается концепция построения главной программы, выполняющей широкие функции путем вызова модулей, реализующих специализированные задачи, причем главная программа и реализующие модули могут быть написаны на разных языках программирования.

Основой разработки служит практический опыт, накопленный при обслуживании матобеспечения машины ZAM-41 и в ходе работ по созданию библиотеки матобеспечения в системе OS/360.

CONCEPT OF THE INTEGRATED UTILITIES

Summary

A very important problem arising in establishing a library of system utilities is their number and variety of ways of their use while their actions are often similar.

The paper presents the example of integrated service programs allowing their reduction to only a few as well as standardization of control. The concept of main program design is discussed. The program plays a very important part by calling modules which perform specialized functions.

The main program and function modules can be written in various programming languages. The paper is based on practice with the service of ZAM 41 computer and with the establishing of software library of the OS/360 system.

ALGORYTMY, Zeszyt Specjalny, 1974, 193-202

681.322.06:
061.3/438/"1973"

ORGANIZACJA VOLUMINÓW DOKUMENTACYJNYCH

Jerzy SWIANIEWICZ
Marian SKUPIŃSKI

Zakład Doświadczalny Oprogramowania
Instytutu Maszyn Matematycznych

Pracę złożono 10.I.1974

Przedstawiono metodę dokumentowania kolejnych wersji oprogramowania wraz z formą określania ich edycji i podstawowego opisu. Zapewnia ona możliwość zidentyfikowania postaci źródłowej oprogramowania na podstawie informacji włączanych do wynikowej postaci oprogramowania, jak również eliminuje możliwość powstania rozbieżności między dokumentacją oprogramowania a stanem faktycznym. Podstawą opracowania stały się praktyczne doświadczenia wynikające z pracy przy serwisie oprogramowania maszyny ZAM 41 oraz w trakcie tworzenia biblioteki oprogramowania w systemie OS/360.

S p i s   t r e ś c i

WSTĘP

W opracowaniu opisano pewną koncepcję organizacji biblioteki oprogramowania możliwą do realizacji na maszynach cyfrowych średniej wielkości. Koncepcja ta jest wynikiem doświadczeń zebranych przy serwisie oprogramowania maszyny ZAM 41 oraz przy realizacji biblioteki oprogramowania w systemie OS/360.

Celowo pominięto pewne szczegóły konkretnej realizacji dla podkreślenia ogólności opisanej organizacji.

1. MATERIAŁ ŹRÓDŁOWY OPROGRAMOWANIA

Materiał źródłowy oprogramowania (MZO) - jest to ujednolicony i zapisany na maszynowych nośnikach informacji (na ogół taśmach magnetycznych), zespół dokumentów opisujących zarówno w sensie ideowym jak i dokumentacyjnym (faktycznym) określony system oprogramowania. Wymaga się ponadto, aby dla konkretnego MZO była ściśle opisana technika jego aktualizacji oraz procedura przechodzenia od MZO do postaci wynikowej oprogramowania. Przez postać wynikową oprogramowania rozumiemy taką jego postać w jakiej musi się ono znajdować w maszynie podczas eksploatacji systemu. W następnych paragrafach opisano realizację biblioteki MZO wykorzystującą jako podstawowy nośnik taśmę magnetyczną.

2. STRUKTURA MATERIAŁU ŹRÓDŁOWEGO OPROGRAMOWANIA

MZO zawarty jest na voluminach źródłowych (VZ). Voluminami źródłowymi są szpule taśmy magnetycznej, a zapisane na niej pliki nazywamy plikami źródłowymi (PZ).

Pliki źródłowe są zbiorami dokumentów tworzących możliwie pełną dokumentację elementów oprogramowa-

nia (EO). Cechą elementu oprogramowania jest posiadanie postaci wynikowej.

Oprócz właściwych plików źródłowych volumin źródłowy może zawierać również tzw. fikcyjne pliki źródłowe zawierające jedynie dokumentację opisową oprogramowania i nie pozostawiające śladu w postaci wynikowej. Do tego typu plików należy pierwszy plik każdego voluminu źródłowego. Jest to plik o nazwie "DATA". Służy on do ewidencjonowania zmian wprowadzanych do voluminu źródłowego oraz jest nośnikiem edycji voluminu źródłowego (EVZ). Edycją voluminu źródłowego jest tekst trzyliterowy. Zwiększenie EVZ następuje zawsze w wypadku dokonania zmian w voluminie źródłowym.

Zawartość pliku źródłowego podzielona jest na strony.

Strona jest to ciąg dokumentów pliku zaczynający się od dokumentu, zwanego znacznikiem strony. Budowa znacznika strony jest zależna od konkretnej realizacji oprogramowania biblioteki MZO. Końcem strony jest początek strony następnej lub koniec pliku.

W pliku źródłowym wyróżniamy następujące części, z których każda składa się z całkowitej liczby stron (niektóre z tych części mogą być puste):

- strona ewidencji zmian PZ
- postać źródłowa EO
- procedury sterowania zadaniami (JCL - Job Control Language)
- test postaci wynikowej EO (dane, program)
- opis funkcjonalny EO
- opis realizacji EO

Strona ewidencji zmian PZ spełnia w nim funkcję podobną jak pierwszy plik voluminu źródłowego, a więc zawiera informacje o dokonanych zmianach w PZ i jest nośnikiem edycji pliku źródłowego (EPZ). Edycja pliku źródłowego jest parą elementów trójznakowych, z których pierwszy podaje edycję voluminu źródłowego jaki powstał podczas ostatnich

poprawek danego pliku źródłowego, a drugi jest liczbą trzycyfrową wskazującą ile razy ten plik był poprawiany. Drugi człon EPZ jest więc zwiększany o 1 w przypadku dokonywania zmian wewnątrz pliku źródłowego. Ten drugi człon nazwiemy l i c z n i k i e m z m i a n P Z (LPZ). Można więc zapisać budowę EPZ:

<EPZ> ::= <EVZ> - <LPZ>

Przykłady: AAB - 001, ABD - 015.

Strony wchodzące w skład pliku źródłowego posiadają e d y c j ę s t r o n y (EST) skonstruowaną analogicznie do edycji pliku źródłowego:

<EST> ::= <EPZ> - <LST>

gdzie: pierwszy człon jest edycją pliku źródłowego, który powstał podczas wprowadzania ostatnich zmian na danej stronie, drugi człon jest licznikiem zmian na stronie.

Przykłady edycji stron:

AAA - 000 - 000, AXY - 029 - 013

Należy tu zauważyć, że o ile ostatni człon edycji strony zmienia się w sposób regularny (o 1 przy każdej zmianie strony), o tyle wyższe człony edycji zmieniają się nieregularnie ze względu na częstsze, na ogół, zmiany edycji elementów nadrzędnych (plików, voluminów). Istnieje zasada, że przy dokonywaniu zmian w voluminie (pliku) edycje niezmienianych plików (stron) nie zmieniają się.

Edycja strony zawarta jest w dokumencie rozpoczynającym stronę (znacznik strony). Dokument ten zawiera również znak klasyfikujący stronę jako element jednej z wymienionych części pliku źródłowego.

Element oprogramowania zawarty w pliku źródłowym ma również przyporządkowaną edycję. Przyjmuje się, że e d y c j a e l e m e n t u o p r o g r a m o w a n i a (EEO) równa się najwyższej wartości drugiego członu (tzn. licznika zmian PZ) edy-

cji stron tworzących postać źródłową elementu oprogramowania. Edycja elementu oprogramowania winna być tak wmontowana w postać źródłową tego elementu, aby zawsze przechodziła do jego postaci wynikowej. Sposób wmontowania EEO może być różny dla różnych typów EO.

Postać źródłowa EO jest to formalny zapis elementu oprogramowania przetwarzany za pomocą odpowiednich środków programowych (np. translatorów odpowiednich języków) na postać wynikową EO.

Procedury sterowania zadaniami (JCL) wchodzące w skład pliku źródłowego opisują różnego rodzaju transformacje wykonywane na elemencie oprogramowania, szczególnie sposób przekształcania postaci źródłowej tego elementu na jego postać wynikową. Dopuszcza się dowolną liczbę takich procedur w jednym pliku źródłowym.

Opis funkcjonalny przeznaczony jest dla potencjalnego użytkownika EO i powinien zawierać informacje opracowane pod tym kątem.

Opis realizacji przeznaczony jest dla programisty wprowadzającego zmiany do istniejącego EO. Winien tu być zawarty opis dokładnej budowy wewnętrznej elementu oprogramowania i ewentualnie współdziałanie między poszczególnymi częściami elementu.

Test postaci wynikowej EO określa metodę sprawdzenia poprawności nowo otrzymanej wersji elementu oprogramowania. Istotnym wymaganiem jest nadanie tej części pliku źródłowego postaci umożliwiającej automatyczne sprawdzenie poprawności EO bez udziału programisty.

3. BUDOWA PLIKÓW ŹRÓDŁOWYCH DLA WYRÓŻNIONYCH KLAS ELEMENTÓW OPROGRAMOWANIA

Pewne szczegóły budowy plików źródłowych są różne dla różnych typów elementów oprogramowania. Wprowadzimy zatem podział

elementów oprogramowania na klasy z punktu widzenia organizacji ich dokumentacji.

Poniżej będą omówione przykładowe dwie klasy elementów oprogramowania:

- klasa TEKSTÓW
- klasa PROGRAMÓW

Należy się spodziewać, że w miarę rozwoju prac dokumentacyjnych będą się pojawiać nowe kategorie elementów oprogramowania, dla których trzeba będzie określać konkretne struktury plików źródłowych. Winny one jednak zawsze spełniać wymagania ogólne określone w p. 2.

3.1. Teksty

Tekstami nazywamy te elementy oprogramowania, których postać wynikowa jest zbudowana z dokumentów stanowiących postać źródłową EO w pliku źródłowym. Można więc powiedzieć, że ta klasa elementów oprogramowania ulega przekształceniu tożsamościowemu podczas przechodzenia z postaci źródłowej do wynikowej. Zmianie ulega jedynie organizacja zbioru, z sekwencyjnego pliku źródłowego na bibliotekę dostosowaną do wymagań konkretnego systemu operacyjnego dla postaci wynikowej.

Obecnie nie można określić szczegółowych wymagań dla wszystkich możliwych rodzajów tekstów. Zasadniczo żąda się, aby każdy z nich zawierał informacje o własnej edycji podaną w sposób umożliwiający użytkownikowi identyfikację, tzn. bez odwoływania się do postaci źródłowej.

W następnym punkcie podane są przykładowe wymagania dla tekstów będących m a k r o r o z k a z a m i w języku ASSEMBLER/360.

3.1.1. Makrorozkazy języka ASSEMBLER/360

Każdy z makrorozkazów posiada nazwę, będącą jednocześnie nazwą elementu zawierającego ten makrorozkaz w określonej bibliotece.

W trakcie rozwijania makrorozkazu winno być zapewnione wygenerowanie zdania, które jest komentarzem w języku ASSEMBLER, podającego edycję makrorozkazu.

W części opisowej, niezależnie od zagadnień funkcjonalnych powinny być określone następujące informacje:

- nazwa makrorozkazu
- biblioteka - nazwa zbioru typu partitioned, w którym znajduje się postać wynikowa makrorozkazu
- typ - informacja określa, do którego z typów (łącznik, pieczątka, tablica, podprogram) zaliczany jest makrorozkaz.

Wśród procedur sterowania zadaniami (JCL) musi wystąpić procedura określająca sposób skopiowania postaci źródłowej EO do jej postaci wynikowej. Oprócz tej procedury dla makrorozkazów typu "podprogram", mogą wystąpić dodatkowo procedury określające różne sposoby generowania pewnych szczególnych wersji podprogramu, przechowywanych w bibliotece w postaci modułów typu "load". W takim przypadku szczegółowe informacje o parametrach generowania oraz o bibliotekach, w których te moduły i ich nazwy będą umieszczone, winny być zawarte w części opisowej odpowiednich elementów oprogramowania.

3.2. Programy

Programami nazywamy te elementy oprogramowania, których postać wynikową uzyskujemy po przekształceniu ich za pomocą translatorów języków programowania. Postać wynikowa może być umieszczana w bibliotece dostosowanej do wymagań systemu operacyjnego lub wyprowadzana na odpowiednich nośnikach danych. Ogólnym wymaganiem dla tej klasy EO jest posiadanie in-

formacji, zawartych w postaci wynikowej, określających nazwę EO i jego edycję.

Sposób umieszczania tych informacji w postaci wynikowej, jak również dodatkowe wymagania dokumentacyjne w odniesieniu do pozostałych części pliku źródłowego są uzależnione od użytego języka programowania i systemu operacyjnego.

Przykładowo podamy wymagania dla pewnej grupy programów, nazwanej m o d u ł a m i p r o s t y m i, istniejącej w systemie operacyjnym OS/360.

3.2.1. Moduły proste

Moduły proste są to programy napisane w dowolnym języku programowania. Ich postacią wynikową jest m o d u ł t y p u "l o a d" będącego elementem określonej b i b l i o t e k i (tzn. skatalogowanego zbioru typu "partitioned"). Z każdym z takich programów związane są dwa następujące symbole:

a) n a z w a b i b l i o t e c z n a – zmienna określająca punkt wejścia do programu oraz będąca nazwą elementu biblioteki zawierającego postać wynikową tego elementu oprogramowania. Symbol ten pozostaje niezmienny przy kolejnych aktualizacjach programu,

b) z m i e n n a e d y c y j n a – symbol 8 znakowy, w którym 5 pierwszych znaków określa s y m b o l e w i d e n c y j n y elementu oprogramowania, zaś 3 ostatnie (będące cyframi dziesiętnymi) – e d y c j ę e l e m e n t u. Ta druga część symbolu zmienia się więc przy każdej poprawce postaci źródłowej EO.

Edycja elementu (liczba wyznaczona przez 3 ostatnie znaki zmiennej edycyjnej) jest również "wmontowana" w sam element w postaci pierwszego rozkazu modułu, którym jest rozkaz NOPx, gdzie x jest edycją EO.

W części opisowej modułów prostych podane są następujące ich charakterystyki:

- t y p. Typem może być p r o g r a m - gdy moduł jest wywoływany samodzielnie (za pomocą zdania EXEC), lub p o d-p r o g r a m - gdy nie jest przystosowany do tego, aby być sterowanym przez zdania JCL,

- p o s t a ć b i b l i o t e c z n a. Może być określona jako m o d u ł o t w a r t y - gdy moduł typu "load" posiada nie rozwiązane odwołania zewnętrzne i jego wykorzystanie musi być poprzedzone opracowaniem go przez Linkage-Editor. Do takiego modułu nie można szczególnie odwoływać się za pomocą makrorozkazów LINK, XCTL,

- wartość m o d u ł z a m k n i ę t y oznacza, że w postaci wynikowej moduł może być wykonywany. Posiada "rozwiązane" wszystkie odwołania zewnętrzne i można odwoływać się do niego za pomocą makrorozkazów LINK, XCTL,

- b i b l i o t e k a - określa nazwę zbioru partitioned, w którym znajduje się postać wynikowa tego elementu.

We wszystkich modułach prostych zmienna edycyjna jest zmienną zewnętrzną programu, a więc może być odczytana z wydruku produkowanego przez Linkage-Editor. W modułach typu program, których w normalnym wykorzystywaniu nie podaje się do przetwarzania Linkage-Editorowi, zmienna edycyjna jest również dodatkową nazwą modułu. Może więc być również odczytana z wydruku directory odpowiedniej biblioteki.

Moduły proste zawierają jedną procedurę sterowania zadaniami (JCL). Jest ona zbiorem zdań JCL określającym sposób przetwarzania postaci źródłowej EO na jego postać wynikową.

ОРГАНИЗАЦИЯ ФАЙЛОВ МАТОБЕСПЕЧЕНИЯ

Резюме

Существенной проблемой при создании больших систем, в том числе и операционных систем, является обоснование очередных вариантов системы и ее элементов. В разработке представляется метод хранения очередных вариантов матобеспечения, включая форму определения их изданий и основного описания. Этот метод позволяет идентифицировать исходный вид матобеспечения на основе информации, включаемой в окончательный вид программ, а также исключает возможность появления расхождения между документацией матобеспечения и фактическим состоянием. Основой труда служит практический опыт, накопленный при обслуживании матобеспечения машины ZAM-41 и в ходе работ по созданию библиотеки матобеспечения в системе OS/360.

THE ORGANIZATION OF SOFTWARE DOCUMENTATION VOLUMES

Summary

While creating large software systems, including the operating systems the documentation of successive versions of such a system and its elements is a vital problem. This paper presents a method of storing the successive versions of the software together with a form of their edition identification and basic description. The method enables identification of source code version from which given module was obtained.

The method gives also a possibility of elimination of controversions between the software documentation and the software it describes.

The paper bases on practice obtained during the service of ZAM 41 software and on the creation of software library on the OS/360 system.

V. PRZYKŁADY SYSTEMÓW OPERACYJNYCH

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

ALGORYTMY, Zeszyt Specjalny, 1974, 205-215

681.322.06:
061.3/438/"1973"

WIELODOSTĘPNY SYSTEM KONWERSACYJNY MACS
Część I. Projekt systemu operacyjnego

Jerzy KARCZEWSKI
Centrum Obliczeniowe PAN

Pracę złożono 10.I.1974

W opracowaniu przedstawiono schematy blokowe najważniejszych fragmentów systemu operacyjnego wchodzącego w skład systemu MACS (Multi-Access Conversational System), który jest eksperymentalnym wielodostępnym systemem konwersacyjnym dla maszyny ODRA 1204. Omówiono również sposób pracy systemu operacyjnego przy narzuconym determinizmie. System MACS realizuje wielodostępność wykorzystując system przerwań zarówno układowych jak i programowych. Omawiając system przerwań szczególną uwagę poświęcono wpływowi poszczególnych zdarzeń na tryb pracy systemu. W zakończeniu omówiono wpływ specyfiki maszyny ODRA 1204 oraz przyjętej konfiguracji systemu na projekt systemu operacyjnego.

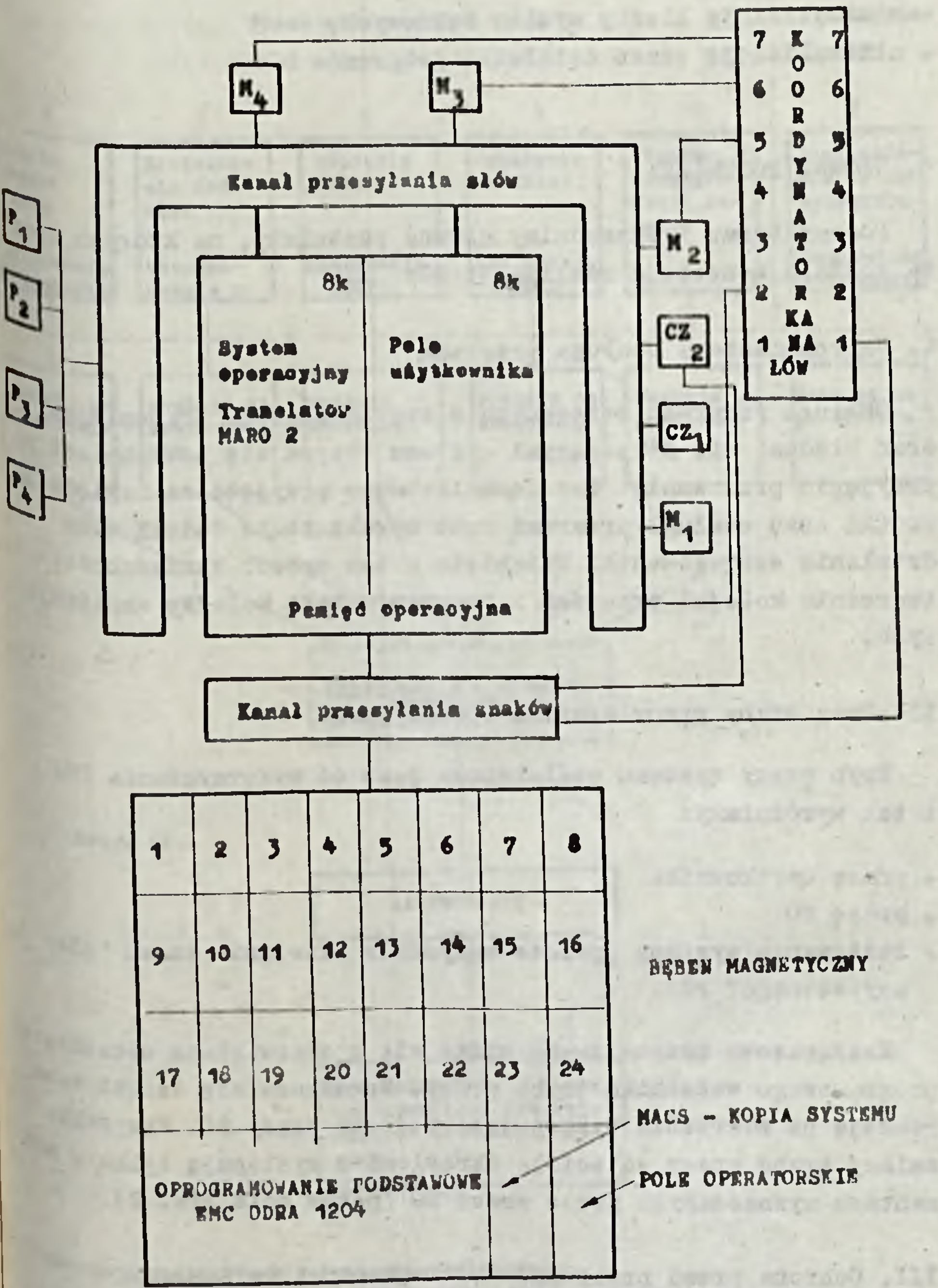
Spis treści

1. WSTĘP

MACS - Multi Access Conversational System jest eksperymentalnym wielodostępnym systemem konwersacyjnym dla maszyny ODRA 1204. System ten, zrealizowany w Centrum Obliczeniowym PAN, umożliwia jednoczesną pracę czterem użytkownikom. Każda stacja wyposażona jest w monitor i perforator, a dwie z nich posiadają ponadto czytnik taśmy papierowej. System operacyjny (SO) współpracuje z "czystym" translatorem języka konwersacyjnego MARO-2. Na rys. 1 przedstawiona została konfiguracja systemu.

2. ZAŁOŻENIA

Chcemy tu przedstawić projekt systemu operacyjnego dla systemu MACS, a dokładniej mówiąc część dotyczącą programów nadzorczych. Zanim jednak przystąpimy do omówienia zasad działania SO, zastanowimy się nad tymi cechami maszyn cyfrowych ODRA 1204, które wpłynęły na koncepcję SO. ODRA 1204 nie jest w zasadzie przewidziana do wykorzystywania w systemach wielodostępnych głównie z uwagi na zbyt małą pojemność pamięci operacyjnej, uniemożliwiającą podział jej między przynajmniej dwóch użytkowników. Maksymalne pole jakie można by przydzielić przy założeniu jednoczesnego podziału pamięci między co najmniej 2 użytkowników wyniosłoby 4 K. Pole takie uznaliśmy za zbyt małe. Przyjęliśmy więc zasadę, że w danej chwili w pamięci operacyjnej (PO) znajdować się może pole jednego użytkownika o wymiarze 8 K. Głównym problemem stało się więc, wobec konieczności dokonywania wymian bębnowych po każdej zmianie przydziału procesora centralnego, zminimalizowanie strat czasowych z tym związanych. Przez wymianę bębnową rozumieć będziemy odesłanie pola użytkownika, któremu skończył się przydział procesora centralnego (PC) do pamięci bębnowej (PB) i przeniesienie z PB do PO pola użytkownika, któremu zostanie przydzielony PC. Sam czas wymian bębnowych jest znaczny (średni czas oczekiwania na obrót bębna) wynosi 20 ms, szybkość transmisji - 12000 słów/s. Dlatego minimalizacja czasu reakcji systemu dokonana została z uwagi na:



M_1, M_2, M_3, M_4 - monitory P_1, P_2, P_3, P_4 - perforatory CZ_1, CZ_2 - czytniki

Rys. 1. Konfiguracja systemu

- minimalizację liczby wymian bębnowych,
- minimalizację czasu działania programów SO.

3. GŁÓWNE POSTULATY

Podamy teraz i uzasadnimy główne postulaty, na których oparta została koncepcja realizacji SO:

I. Natychmiastowa obsługa przerwań

Obsługa przerwań związanych z szeregowaniem, transmisjami oraz błędami nie niszczącymi systemu odbywa się natychmiast po przyjęciu przerwania. Rozwiązania takie przyjęto ze względu na krótki czas obsługi przerwań oraz uproszczenie dzięki temu działania szeregowania. Uniknięto w ten sposób konieczności tworzenia kolejki przerwań i programów taką kolejkę analizujących.

II. Trzy tryby pracy systemu operacyjnego

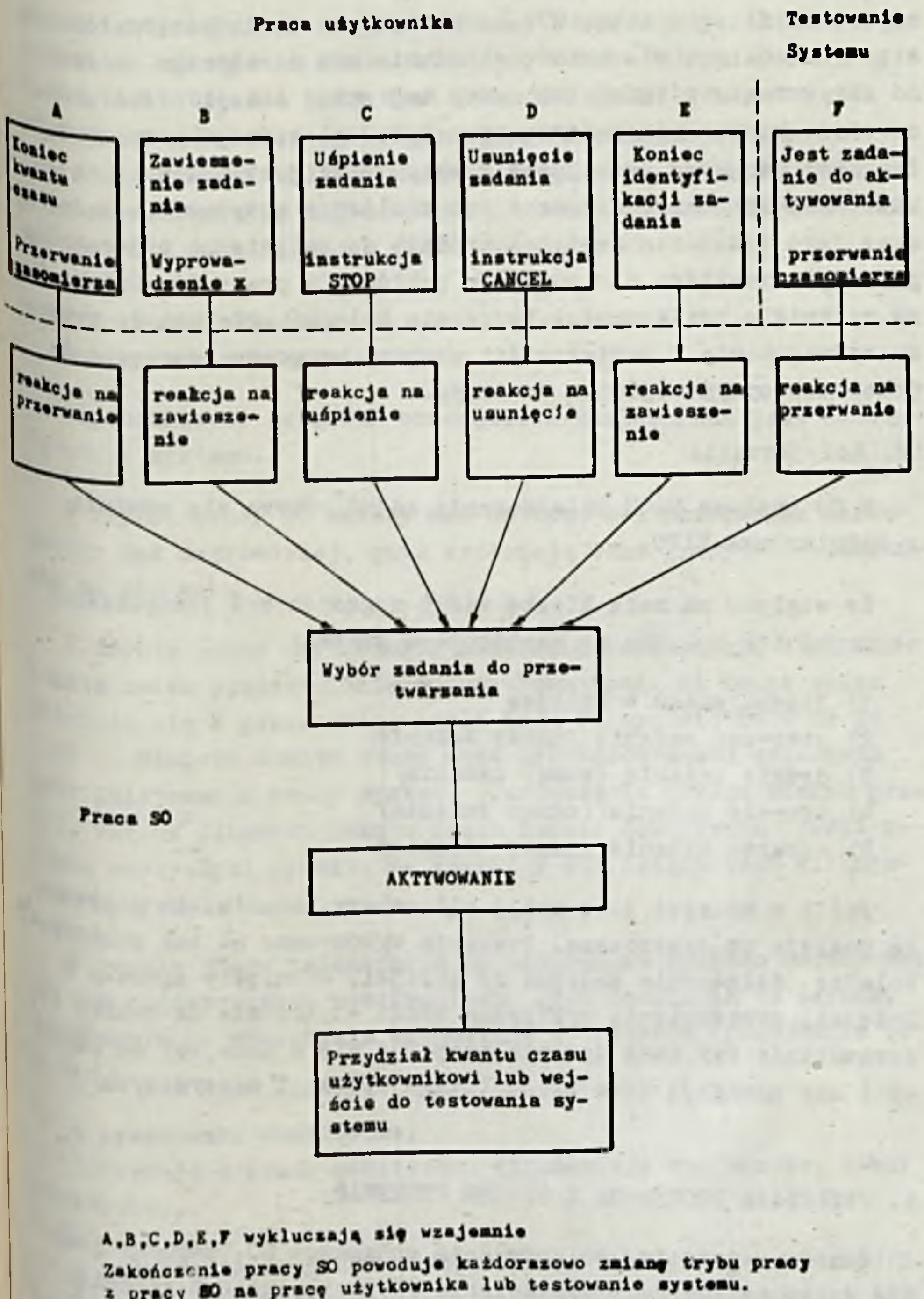
Tryb pracy systemu uzależniony jest od wykorzystania PC, i tak wyróżniamy:

- pracę użytkownika
- pracę SO
- testowanie systemu (przede wszystkim zliczanie czasu "nieużytecznego" PC).

Każdorazowa zmiana trybu wiąże się z odpowiednim ustawieniem programowego wskaźnika trybu pracy. Upraszcza się dzięki temu reakcje na zdarzenia występujące podczas pracy SO. Wszystkie zmiany trybu pracy są ściśle określone i występują tylko w momentach wyznaczanych przez pracę SO (patrz opis rys. 2).

III. Ochrona przed przerwaniami fragmentu programu szeregowania realizującego aktywowanie zadań

Zdarzenia związane ze zgłoszeniem się zadań do aktywowania (np. naciśnięcie klawisza ZO na pulpicie modułu monitora przez



Rys. 2. Ogólny schemat działania SO

użytkownika) są losowe. W celu uniknięcia możliwości nałożenia się tych zdarzeń w czasie (zgłoszenie się następnego zadania do aktywowania z innej lub nawet tej samej stacji przed ukończeniem aktywowania zadania uprzednio zgłoszonego) odpowiedni fragment programu szeregowania wykonywany jest pod ochroną przed przerwaniami. Fragment ten realizuje aktywowanie zadania, czas jego wykonania jest tak krótki, że wzięcie go pod ochronę przed przerwaniami nie powoduje zakłóceń w pracy SO. Dzięki temu uniknięto konieczności tworzenia kolejki zgłoszeń do programu szeregowania i konieczności ochrony programu szeregowania przed następnym wejściem do niego.

IV. Kolejkowanie

W SO systemu MACS kolejkowanie zadań odbywa się zgodnie z regulaminem FIFO.

Ze względu na małą liczbę zadań mogących być jednocześnie czynnymi (4) kolejka ma następującą postać:

1) liczba zadań w kolejce
2) pierwsze zadanie (numer zadania)
3) drugie zadanie (numer zadania)
4) trzecie zadanie (numer zadania)
5) czwarte zadanie (numer zadania)

Jeśli w kolejce jest mniej niż cztery zadania, to pozostałe pozycje są wyzerowane. Operacje wykonywane na tak zbudowanej kolejce (dołączenie zadania do kolejki, usunięcie zadania z kolejki, przesunięcia cykliczne zadań - pierwsze na koniec i sprawdzanie czy dane zadanie znajduje się w kolejce) są proste i nie wymagają dużo czasu (kilka operacji maszynowych).

4. PRZYDZIAŁ PROCESORA I SYSTEM PRZERWAŃ

Zanim przystąpimy do omówienia schematów SO, zwróćmy jeszcze uwagę na zasadę przydzielania PC poszczególnym zadaniom oraz system przerwań.

Jak już powiedzieliśmy jednym z ważniejszych problemów jest zminimalizowanie liczby wymian bębnowych, pod warunkiem, że

nie wydłuży to czasu reakcji systemu. Przyjęliśmy następującą zasadę: udostępniamy użytkownikowi PC tak długo, jak długo nie wpływa to ujemnie na czas reakcji systemu. Polega to na tym, że SO zachowuje się cały czas tak, jak gdyby było czterech użytkowników w kolejce. Drugim ważnym aspektem jest system przerwań. Wykorzystaliśmy zarówno przerwania układowe jak i programowe. Omówimy najważniejsze z nich, wskazując jednocześnie na rolę jaką pełnią w funkcjonowaniu SO.

1. Przerwania układowe
   • przerwanie czasomierza,

Częstotliwość przerwań czasomierza uzależniona jest od trybu pracy systemu.

W trybie pracy SO zależy nam na tym, aby przerwania następowały jak najrzadziej, gdyż wydłużają czas pracy SO (przerwania co 640 ms).

W trybie pracy użytkownika przerwania wyznaczają długość kwantu czasu przydzielania PC użytkownikowi. Na kwant czasu składają się 2 przerwania, czyli może on wynosić od 4 ms do 1.28 s. Długość kwantu czasu może być każdorazowo zmieniona przy inicjowaniu pracy systemu (rozpoczęcie nowego seansu pracy). Zmiana długości kwantu czasu będzie dokonywana, jeśli zebrane statystyki wykażą, że zwiększy się dzięki temu efektywność systemu.

W trybie pracy polegającym na testowaniu systemu zależy nam na jak najczęstszych przerwaniach, gdyż umożliwia to szybkie reagowanie na wydarzenia zachodzące w systemie (przerwanie co 2 ms).

• przerwania monitorowe,
Informują o pracy monitorów (transmisje monitorowe, błędy monitorów),

• przerwania urządzeń zewnętrznych: czytniki i perforatory,
Informują o pracy urządzeń zewnętrznych (transmisje i błędy urządzeń),

- przerwania związane z pracą bębna magnetycznego,

Informują o przebiegu transmisji bębnowych oraz o błędach bębna.

2. Przerwania programowe

- zawieszenie zadania,

Zawieszenie zadania następuje w wyniku wykonania kolejnego zdania programu użytkownika. Zadanie jest zawieszone od momentu wypisania przez SO znaku "x" na monitorze. Jest to stan oczekiwania na wprowadzenie przez użytkownika następnego zdania.

- uśpienie zadania,

Uśpienie zadania następuje w wyniku wykonania przez SO instrukcji STOP wprowadzonej przez użytkownika.

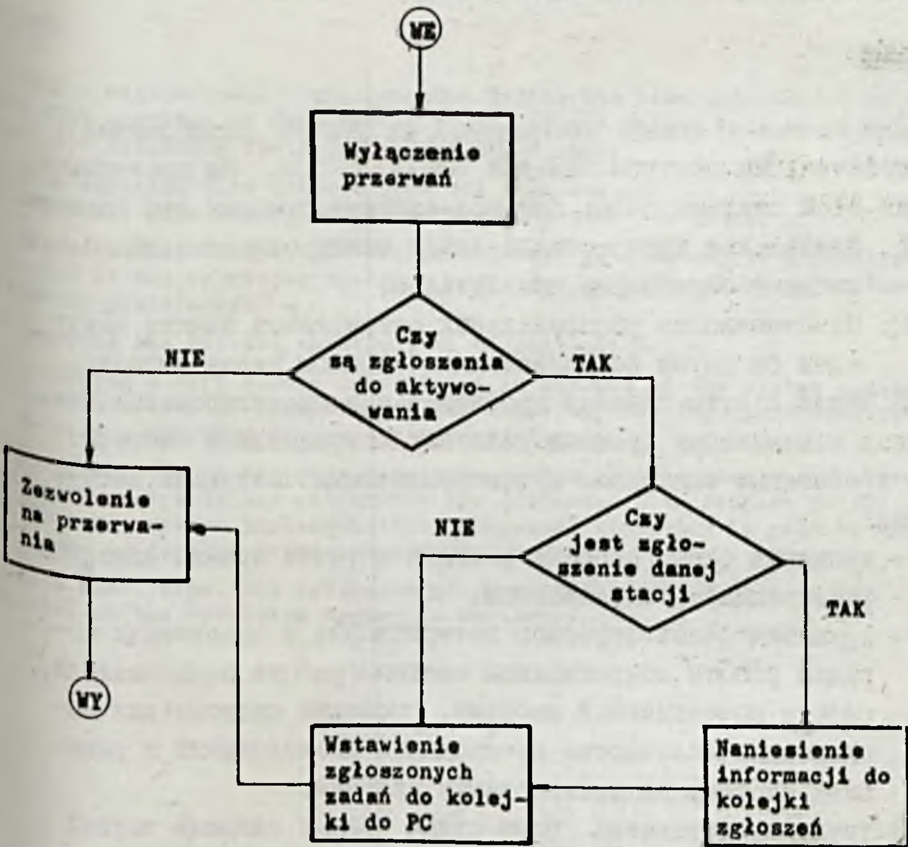
- usunięcie zadania,

Usunięcie zadania następuje w wyniku wykonania przez SO instrukcji CANCEL wprowadzonej przez użytkownika.

5. SCHEMATY SYSTEMU OPERACYJNEGO

Na rys. 2 przedstawiono schemat działania SO. Warto podkreślić, że wejście do programów SO może nastąpić w wyniku wzajemnie wykluczających się sytuacji. Testowanie wyklucza pracę użytkownika. Zawieszenie, uśpienie, usunięcie wykluczają się wzajemnie, jak również wykluczają możliwość dojścia do końca kwantu czasu. Wejście do programów systemu operacyjnego po zakończeniu identyfikacji związane jest z koniecznością przygotowania pola dla użytkownika rozpoczynającego pracę w MARO-2. Sama identyfikacja nie wymaga przyporządkowywania użytkownikowi pola. Postulat minimalizacji liczby wymian bębnowych oraz determinizm SO osiągnięto przez aktywowanie zadania tylko w jednym określonym momencie wykonywania programów systemu operacyjnego. Sprawia to, że moment dołączenia zadania do kolejki zadań czekających na PC jest opóźniony w stosunku do momentu zgłoszenia się. Jednak z punktu widzenia użytkownika opóźnienie to nie występuje, gdyż dołączenie wykonywane jest zanim

mógłby on otrzymać przydział PC. Na zakończenie podany jest schemat aktywowania (rys. 3).



Rys. 3. Aktywowanie

6. ZAKOŃCZENIE

Przyjęta konfiguracja systemu oraz specyfika EMC ODRA 1204 wpłynęły w zasadniczy sposób na projekt systemu operacyjnego. Omawiając projekt systemu dużo uwagi poświęciliśmy sposobowi praktycznego zminimalizowania konsekwencji wynikających z ograniczeń sprzętowych. Do systemu włączony został bogaty aparat zbierania statystyk, które mamy nadzieję, wykażą słuszność przyjętej konfiguracji i projektu SO z punktu widzenia efektywności systemu.

СИСТЕМА С РАЗДЕЛЕНИЕМ ВРЕМЕНИ ТИПА "ВОПРОС-ОТВЕТ" MACS
ЧАСТЬ I. ПРОЕКТ ОПЕРАЦИОННОЙ СИСТЕМЫ

Резюме

MACS - Multi Access Conversational System разработана Вычислительным центром ПАН для ЭВМ ODRA-1204. На проектирование этой системы очень большое влияние оказали два фактора:

1. Длительное время обмена задач между оперативной памятью и барабанным накопителем;
2. Невозможность распределения оперативной памяти между хотя бы двумя пользователями(слишком малая емкость).

В связи с этим главной проблемой при проектировании оказалась минимизация времени действия операционной системы. Это требование выполнено путем реализации следующих постулатов:

- принятия самых простых решений с точки зрения машинной реализации(время действия);
- принятия очень строгого детерминизма в изменениях порядка работы операционной системы(работа пользователя, работа операционной системы, проверка системы) для минимизации количества необходимых исследований в реакциях системы на происходящие явления.

В труде представлена блок-схема самых главных частей операционной системы. Операционная система MACS реализует разделение времени на основе системы технических и программных прерываний.

При обсуждении проблемы прерываний особенное внимание уделяется влиянию отдельных явлений на порядок работы операционной системы.

В заключение обсуждается влияние специфических свойств машины ODRA-1204 и принятой конфигурации на проект операционной системы.

A MULTIACCESS CONVERSATIONAL SYSTEM MACS
Part I. THE OPERATING SYSTEM DESIGN

Summary

MACS - Multi-Access Conversational System has been implemented on the ODRA 1204 computer in Computation Centre of the Polish Academy of Science. Two following facts influenced the design:

1) Long exchange time between core and drum memory,
2) Too small core memory to be shared among even two users.

These factors caused that the main design problem was to minimalize the time of the operating system's work. The solution is based on two following postulates:

1) choosing the fastest solutions to be implemented,
2) enforcing a very strong determinizm in changes of the system's modes (user's activity, operating system activity, testing) to minimalize necessary scanning in the system activity.

Flow diagrams of the main operating system programs are shown. The hardware interrupts and extracodes are discussed with respect to the multi-access system implementation. A special attention is paid to the influence of various events on the system work.

As a conclusion, the influence of characteristic features of the hardware on the operating system is mentioned.

ALGORYTMY, Zeszyt Specjalny, 1974, 217-232

681.322.06:
061.3/438/"1973"

WIELODOSTĘPNY SYSTEM KONWERSACYJNY MACS
Część II. Obsługa wejść-wyjść

Alina RUSZKOWSKA
Centrum Obliczeniowe PAN
Pracę złożono 10.I.1974

W opracowaniu podano opis funkcjonalny i strukturę systemu MACS, zrealizowanego w Centrum Obliczeniowym PAN na EMC ODRA 1204. MACS może być wykorzystywany w ośrodkach wyposażonych w EMC ODRA 1204 z dwoma lub więcej jednostkami pamięci bębnowej i czterema stacjami, dla których wyposażeniem podstawowym jest monitor, a opcjonalnym perforator i czytnik taśmy papierowej. W opisie struktury systemu szczególną uwagę zwrócono na wpływ ograniczeń sprzętowych na budowę systemu.

S p i s t r e ś c i

1. WSTĘP

Omawiany system bazuje na języku konwersacyjnym MARO-2. MACS pozwala na rozwiązywanie problemów naukowo-technicznych w trybie konwersacyjnym, co jest niezwykle cenne w przypadku wykonywania niewielkich obliczeń. Dwa tryby wykonywania instrukcji języka MARO-2, programowy i natychmiastowy, pozwa-

lają w łatwy sposób pisać, uruchamiać i wykonywać programy użytkowe.

Celem niniejszego opracowania jest opis funkcjonalny i struktura systemu MACS. W pierwszej kolejności omówiona będzie konfiguracja sprzętu.

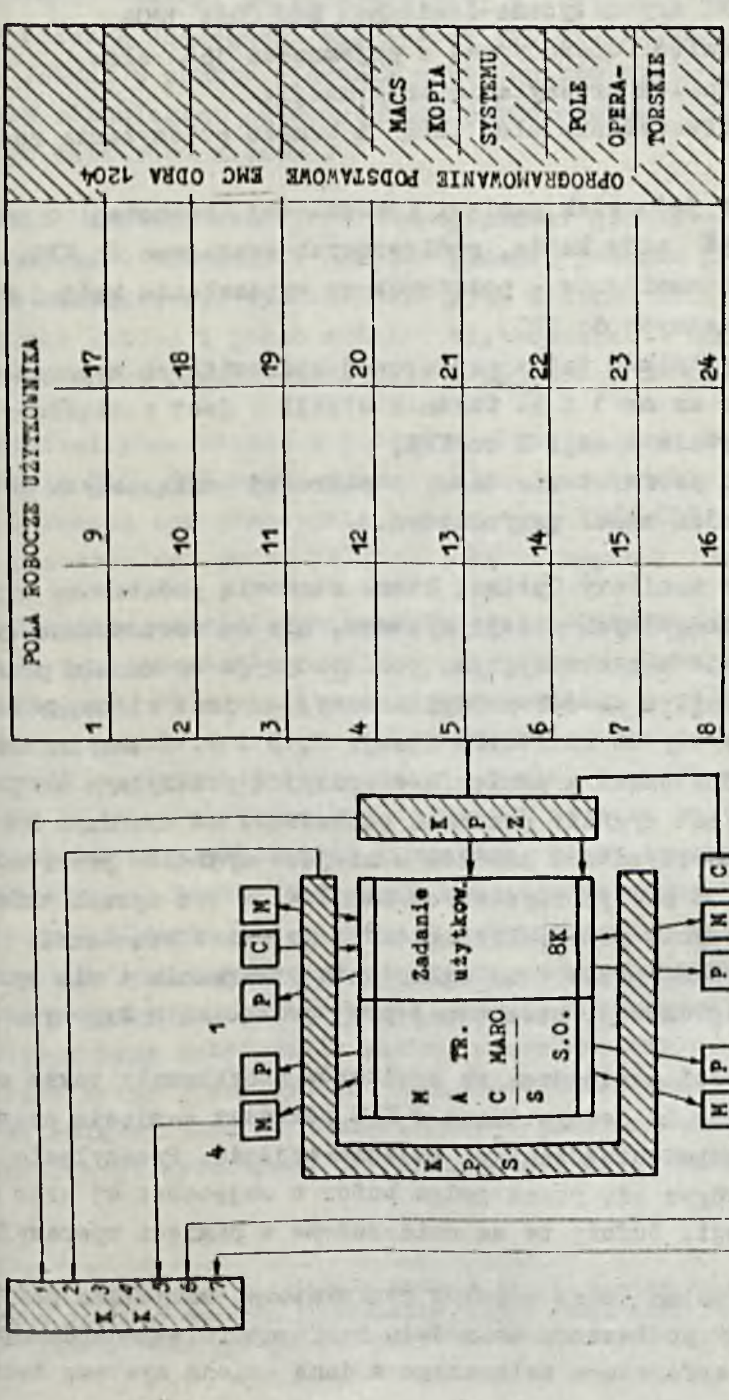
2. KONFIGURACJA SPRZĘTU

Jak powszechnie wiadomo, koszt oprogramowania sprzętu liczącego jest coraz wyższy i w przypadku dużych systemów liczących znacznie przewyższa koszt samego sprzętu. W związku z tym wzrasta znaczenie oprogramowania. Można zaobserwować tendencję do wykorzystywania wykonanego dotychczas oprogramowania przy projektowaniu nowych systemów oprogramowania dla tego samego sprzętu. W naszym przypadku, ze względu na brak oprogramowania EMC ODRA 1204, które dałoby się chociaż fragmentarycznie wykorzystać podczas realizacji systemu MACS, postanowiono przynajmniej nie zmieniać standardowego zestawu sprzętu, aby możliwa była praca w dotychczas wykonanych systemach oprogramowania. W konsekwencji zestaw sprzętu wykorzystywany przez system MACS stanowi rozszerzenie zestawu standardowego o urządzenia wejścia-wyjścia umożliwiające jednoczesną pracę czterem użytkownikom pracującym przy czterech stacjach systemu. Stacją systemu nazywamy zestaw urządzeń wejścia-wyjścia oddanych do dyspozycji użytkownika na okres seansu jego pracy z systemem*. Podstawowym urządzeniem stacji jest monitor, dodatkowym - czytnik i perforator taśmy papierowej. Systemową informację o wyposażeniu stacji w urządzenia dodatkowe aktualizuje operator odpowiednią instrukcją. Ta informacja decyduje o możliwości użycia urządzenia podczas seansu użytkownika.

Zestaw, połączenia i funkcje poszczególnych elementów sprzętu pracującego w Centrum Obliczeniowym PAN obrazuje rys. 1.

---

* Seansem pracy użytkownika nazywamy okres czasu, w którym użytkownik ma dostęp do systemu.



Rys. 1. Konfiguracja systemu MACS

Zestaw składa się z następujących elementów:

1. jednostki arytmetyczno-logicznej EMC ODRA 1204,
2. bloku pamięci operacyjnej o pojemności 16K słów,
3. dwóch kanałów przesyłania informacji: kanału przesyłania słów (KPS) i kanału przesyłania znaków (KPZ),
4. czterech jednostek pamięci pomocniczej (bębnowej) o pojemności 64K słów każda, podłączonych szeregowo do KPZ,
5. czterech monitorów - podstawowego wyposażenia każdej stacji - podłączonych do KPS,
6. dwóch czytników taśmy papierowej stanowiących wyposażenie stacji o nr nr 1 i 2. Czytnik stacji 1 jest podłączony do KPS, czytnik stacji 2 do KPZ,
7. czterech perforatorów taśmy papierowej podłączonych do KPS przez jeden moduł perforatora.

Ponieważ monitory Optima, które stanowią podstawowe wyposażenie poszczególnych stacji systemu, nie są dostosowane do pracy w systemie konwersacyjnym, podłączono je do kanału przesyłania informacji w sposób zmodyfikowany, co jest widoczne na rys. 1. Dotyczy to monitorów stacji 2, 3 i 4. I tak informacje pomiędzy monitorami a pamięcią operacyjną przesyłane są przez KPS, natomiast sygnały przerwań pochodzące od urządzeń są przesyłane do koordynatora kanałów w miejsce sygnałów przerwań pochodzących od niewykorzystanych kanałów. W ten sposób informacja o przyczynie przerwania zgłoszonego przez urządzenie jest rozszyfrowywana przez samo zgłoszenie przerwania i nie wymaga analizy programowej, niezbędnej przy podłączeniu typowym.

Właściwości zastosowanych monitorów podyktowały także zasadę transmisji po jednym znaku w KPS pomiędzy pamięcią operacyjną a wszystkimi urządzeniami wejścia-wyjścia. Przesyłanie informacji odbywa się przez jeden bufor o objętości 41 słów dla każdej stacji. Bufory te są umieszczone w pamięci operacyjnej.

Ze względu na koszt modułów perforatora, wszystkie perforatory zostały podłączone do modułu perforatora standardowego. O numerze perforatora związanego z daną stacją systemu decydu-

je część adresowa maszynowego rozkazu inicjowania transmisji w programach obsługi wejścia-wyjścia. W rzeczywistości perforatory pracują szeregowo, cyklicznie wyprowadzając po jednym znaku. Jednak z punktu widzenia użytkowników praca perforatorów jest równoległa; na wszystkich perforatorach jednocześnie może być perforowana taśma.

Pamięć systemu MACS jest dwupoziomowa: główna - operacyjna i pomocnicza - bębnowa. W pamięci głównej podczas pracy systemu MACS znajduje się system operacyjny systemu MACS, translator języka MARO-2 i jedno zadanie użytkownika. Z tego na zadanie użytkownika przeznaczono pole o objętości 8K słów.

W pamięci pomocniczej w końcowym obszarze przechowywana jest kopia MACS (8K słów) wraz z polem operatorskim (także 8K słów) w ramach oprogramowania podstawowego EMC ODRA 1204[*], na które pozostawiono jedną jednostkę bębna magnetycznego.

W pozostałej części pamięci bębnowej umieszczono 24 pola robocze do przechowywania zadań użytkowników w okresie seansu pracy systemu. Pole operatorskie przeznaczone jest do przechowywania programów operatora, aktualnych informacji o zleceniach użytkowników (dotychczasowe koszty, hasło, stan zlecenia) i informacji o pracy użytkowników (długości seansów użytkowników, "czasy reakcji" użytkowników i systemu, czasy transmisji itp.). W zależności od liczby jednostek bębna magnetycznego jaką dysponuje MACS, liczba pól roboczych przeznaczonych na zadania użytkowników, jak też obszar przeznaczony na oprogramowanie podstawowe mogą być zmieniane przez podanie odpowiednich wartości parametrów przy inicjowaniu seansu pracy systemu przez operatora. Przez seans pracy systemu rozumiemy okres czasu od momentu wprowadzenia systemu do pamięci operacyjnej do momentu wykonania instrukcji systemowej OFF powodującej wyłączenie systemu.

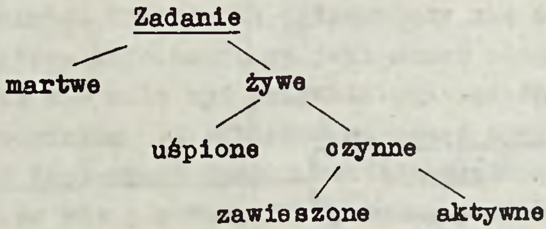
[*] Przez oprogramowanie podstawowe rozumie się tu zestaw programów dostarczonych przez producenta (translator języka ALGOL itp.), umożliwiających wykorzystanie maszyny poza systemem MACS.

3. OPIS FUNKCJONALNY

Inicjowanie seansu pracy systemu odbywa się ze stacji nr 1 (standardowy monitor, czytnik i perforator). System MACS może być wprowadzony do pamięci operacyjnej z pamięci bębnowej lub z taśmy binarnej. Do wprowadzenia systemu służy specjalny program przechowywany na taśmie binarnej. Przeprowadza on wstępną konwersację z operatorem, podczas której operator podaje datę i ewentualnie nowe wartości parametrów systemu, takich jak: współczynniki kosztów wykorzystania procesora i stacji, liczba pól roboczych w pamięci pomocniczej i początkowy adres kopii systemu. Proces inicjowania seansu pracy systemu jest szczegółowo opisany w [2]. Sygnałem rozpoczęcia seansu jest wyprowadzenie uwagi SYSTEM ON. Od tego momentu liczony jest czas globalny pracy systemu, zliczane są koszty pracy użytkowników, a także zbierane są informacje o zadaniach użytkowników. Zakończenie seansu pracy systemu jest wynikiem wykonania instrukcji operatora OFF (patrz [2]), albo wykrycia błędu w pracy sprzętu lub programów, który uniemożliwia dalszą pracę systemu. W przypadku, gdy operator zakończy seans pracy systemu, na monitory wszystkich stacji wyprowadzana jest uwaga SYSTEM OFF, a w przypadku awarii - uwaga SYSTEM ERROR n, gdzie n oznacza numer błędu. Na zakończenie seansu wyprowadzana jest lista wykorzystywanych zleceń z obciążającymi je kosztami oraz informacje o pracy użytkowników zbierane w polu operatorskim.

Jak już wspomniano MACS umożliwia tworzenie, uruchamianie i wykonywanie jednocześnie czterech programów obliczeniowych w języku MARO-2. Ciąg instrukcji języka MARO-2 napisany przez użytkownika na monitorze stacji systemu nazywamy zadaniem. Zadanie użytkownika może znajdować się w jednym z czterech stanów: martwe, uśpione, zawieszone lub aktywne. Zbiór stanów zadania przedstawiono na rys. 2.

Zadanie martwe w odróżnieniu od zadania żywego nie może być wykonywane, ponieważ nie ma żadnego obrazu w pamięci systemu. Zadanie znajduje się w stanie martwym przed pierwszym wywołaniem lub w wyniku zakończenia seansu instrukcją CANCEL.



Rys. 2. Stany zadania użytkownika systemu MACS

Zadanie uśpione zajmuje pole robocze w pamięci pomocniczej, ale nie ma przyporządkowanej żadnej ze stacji systemu. Ostatni seans użytkownika zakończył się instrukcją STOP.

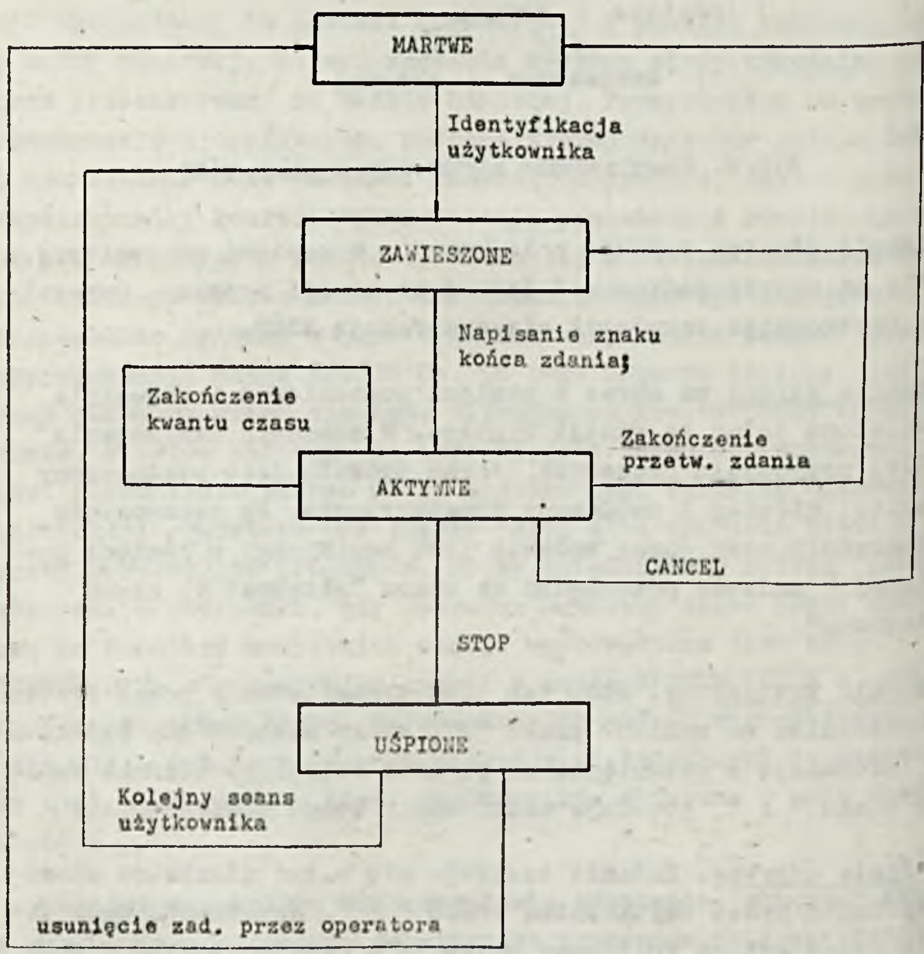
Zadanie czynne ma obraz w pamięci pomocniczej i aktualnie przydzieloną jedną ze stacji systemu. W momencie aktywowania (ściślej przydziału procesora) obraz zadania jest przenoszony do pamięci głównej i poddawany przetwarzaniu. Po zakończeniu przetwarzania nowy obraz zadania jest zapisywany w pamięci pomocniczej - zadanie przechodzi ze stanu "aktywne" do stanu "zawieszone".

Zadanie zawieszone. Stan ten jest sygnalizowany przez system wyprowadzeniem na monitor znaku "X", który stanowi dla użytkownika informację o zezwoleniu na pisanie kolejnego zdania. Napisanie znaku " ; " powoduje zakończenie stanu zawieszenia.

Zadanie aktywne. Zadanie znajduje się w tym stanie od momentu napisania przez użytkownika znaku " ; " do momentu wyprowadzenia przez system kolejnego znaku "X". Jest to jedyny stan, podczas którego zadanie znajduje się w kolejce do procesora.

Zmiany stanów zadania w czasie seansu pracy systemu przedstawiono schematycznie na rysunku 3.

Podczas seansu pracy systemu MACS na liście zleceń, która znajduje się na polu operatorskim w pamięci pomocniczej, przechowywane są informacje o 500 zleceniach ponumerowanych od 0 - 499. Numer 0 jest przyporządkowany na stałe programom operatorskim, które są wywoływane w taki sam sposób jak programy



Rys. 3. Zmiany stanów zadania użytkownika

użytkowników. Użytkownik zgłaszający się do systemu otrzymuje od operatora informację na jaki numer zlecenia może pracować. Numer ten nie może być przydzielony wcześniej innemu użytkownikowi i powinien być odblokowany przez operatora za pomocą odpowiedniej instrukcji operatorskiej. Użytkownik, który zgłasza się nie po raz pierwszy do systemu, powinien znać przydzielony mu numer zlecenia.

W seansie pracy użytkownika można wyróżnić trzy fazy:

1. identyfikacja użytkownika,
2. pisanie dowolnych instrukcji w języku MARO-2 poza instrukcjami STOP i CANCEL,
3. zakończenie seansu instrukcją STOP lub CANCEL.

Każdy użytkownik systemu jest identyfikowany przez numer zlecenia i odpowiadające mu hasło wprowadzone przez użytkownika w poprzednim seansie i znane wyłącznie jemu. W celu ochrony numeru zlecenia przydzielonego danemu użytkownikowi przed obciążeniem go kosztami przez innych, hasło użytkownika jest informacją tajną nawet dla operatora systemu. Przy pierwszym zgłoszeniu się do systemu hasło użytkownika jest puste, to znaczy system akceptuje każde formalnie poprawne hasło. W przypadku nieznajomości hasła użytkownik nie jest dopuszczany do drugiej fazy seansu.

W fazie drugiej użytkownik może tworzyć swój program, modyfikować i wykonywać go. Możliwe jest m.in. wyprowadzenie taśmy binarnej programu w celu późniejszego wprowadzenia go ze stacji wyposażonej w czytnik.

Instrukcje języka MARO-2, obsługa urządzeń stacji i sposób obliczania kosztów wykorzystania systemu zostały opisane w [1].

Bieżące saldo kosztów zlecenia system wyprowadza w fazie identyfikacji i w fazie zakończenia seansu pracy użytkownika. Koszty zlecenia zależą od wyposażenia stacji, czasów pracy procesora i monitora na rzecz zlecenia oraz od czasu przechowywania uśpionego zadania w pamięci pomocniczej.

Na zakończenie fazy trzeciej system wyprowadza uwagę BYE, która oznacza zwolnienie stacji dla następnego użytkownika.

Jak już wspomniano programy operatorskie wywoływane są w ten sam sposób jak zlecenia użytkowników. Instrukcja CANCEL, powodująca usunięcie zlecenia jest traktowana jako niepoprawna. W związku z tym zlecenie O nie przyjmuje stanu "martwe", może być "uśpione" lub "czynne".

Odpowiednie instrukcje operatorskie pozwalają blokować i odblokowywać numery poszczególnych zleceń, otrzymywać informacje o stanie zleceń a nawet usuwać zlecenia uśpione w przypadku przepełnienia systemu tzn. w sytuacji, gdy nie ma pola roboczego w pamięci pomocniczej dla kolejnego użytkownika zgłaszającego się do systemu.

4. STRUKTURA SYSTEMU

Poważne ograniczenia narzucone przez sprzęt miały istotny wpływ zarówno na proces projektowania systemu jak też na produkt, tzn. sam system. Zaważyło to szczególnie na strukturze systemu operacyjnego systemu MACS.

Projekt systemu wykonany w fazie wstępnej, w której poszczególne elementy programowe rozważane były głównie od strony funkcjonalnej, uległ znacznym zmianom w fazie weryfikacji ze względu na możliwości realizacji. Dążenie do skrajnego zminimalizowania czasów działania i objętości zajmowanej pamięci spowodowało konieczność przyjmowania najprostszych z punktu widzenia realizacji rozwiązań. Zrezygnowano także z jednolitości w budowie tablic systemowych na rzecz łatwości dostępu do informacji w nich zawartych.

Programy systemu operacyjnego można podzielić na dwa zbiory:

1. programy obsługi wejścia-wyjścia do komunikacji z urządzeniami zewnętrznymi,
2. programy nadzorcze do efektywnego wykorzystywania zasobów systemu.

Do zadań programów nadzorczych należy szeregowanie zadań użytkowników, podział pamięci pomocniczej pomiędzy te zadania, kontrolowanie pracy urządzeń wejścia-wyjścia i księgowanie kosztów wykorzystania systemu. Programy nadzorcze, ich projektowanie i strukturę omawia J. Karczewski w pierwszej części tego opracowania.

Omówimy tu programy obsługi wejścia-wyjścia ze zwróceniem szczególnej uwagi na ich strukturę. Programy te ze względów funkcjonalnych można podzielić na cztery typy:

1. obsługa transmisji pomiędzy bębnem magnetycznym a pamięcią operacyjną,
2. obsługa transmisji pomiędzy monitorami a pamięcią operacyjną,
3. wprowadzanie informacji z czytnika,
4. wyprowadzanie informacji na perforator.

Ze względu na to, że w pamięci operacyjnej systemu znajduje się tylko jedno zadanie użytkownika, a kopie pozostałych w pamięci bębnowej, po zakończeniu kwantu czasu następuje na ogół wymiana zadań między pamięcią operacyjną a bębnową. Podczas tej wymiany po zainicjowaniu fizycznej transmisji procesor jest bezczynny. W tym czasie mogą jedynie pracować programy realizujące wcześniej zainicjowane transmisje między pamięcią wewnętrzną a urządzeniami poszczególnych stacji.

Programy grupy pierwszej pracują wyłącznie na użytek systemu i czas ich działania obniża efektywność systemu. Z tego powodu podczas realizacji tych programów szczególną uwagę zwrócono na minimalizację czasu ich wykonywania. Programy wymienione w punktach 2, 3 i 4 obsługują zlecenia użytkowników.

Jak już wspomniano monitory stacji 2, 3 i 4 zostały podłączone do kanału przesyłania słów w sposób nietypowy. Sygnały przerwań pochodzące od tych monitorów nie są wprowadzane do kanału, przez który przesyłane są informacje, lecz są przekazywane do koordynatora kanałów do rejestru przerwań na miejsce sygnałów z kanałów 5, 6, 7. Taka organizacja podłączenia moż-

liwa była dzięki temu, że liczba różnych przerwań kanałowych (z jednego kanału) jest równa liczbie różnych przyczyn przerwań zgłaszanych przez moduł monitora. W przypadku innych urządzeń taka równość nie zachodzi.

To nietypowe podłączenie monitorów pozwoliło na pełniejsze wykorzystanie sprzętu, a jednocześnie na skrócenie programów obsługi. Bowiem przyczyna przerwania pochodzącego od urządzenia, która decyduje o sposobie reakcji na przerwanie, jest identyfikowana przez numer zgłoszenia przerwania związany na stałe z określonym miejscem pamięci operacyjnej. Natomiast w przypadku podłączenia typowego identyfikacja przyczyny przerwania następuje w wyniku analizy programowej.

Jest to przykład przeniesienia funkcji systemu operacyjnego na sprzęt. Nietypowe podłączenie monitorów stanowiące niewielką zmianę w konfiguracji sprzętu pełni funkcję programów analizujących przyczynę przerwania i decydujących o wyborze programu reakcji na przerwanie.

Podczas realizacji programów grupy 2 położono szczególny nacisk na minimalizację czasu wykonywania. Dlatego programy te są wykonywane pod ochroną przed przerwaniami i mają najwyższy priorytet programowy. Wykonanie ich może być wstrzymane przez programy reakcji na przerwania o wyższym priorytecie sprzętowym jak przerwania czasomierza, przerwania pochodzące od urządzeń podłączonych standardowo lub monitory stacji o wcześniejszym numerze, a także przez wykonywane w chwili zgłoszenia przerwania fragmenty programów nadzorczych chronione przed przerwaniami. W celu zminimalizowania czasów takich wstrzymań ochronę przed przerwaniami programów nadzorczych zastosowano wyłącznie tam, gdzie była niezbędna. W wyniku tego większość programów nadzorczych może być przerywana przez programy obsługi urządzeń, lecz nie odwrotnie. Rozwiązanie takie zostało podyktowane bardzo ostrym ograniczeniem łącznego czasu reakcji na wszystkie przerwania techniczne jakie mogą się zdarzyć w systemie. Ograniczenie to narzuca konstrukcja monitorów. Przy założeniu, że użytkownik pisze na monitorze z prędkością do 10 znaków/sekundę, czas jaki można zużyć na obsługę wszystkich przerwań wyno-

si 0.1 ms. Należy tu zaznaczyć, że monitor stacji 1, wchodzący w skład standardowego zestawu sprzętu, jest podłączony w sposów typowy, podobnie jak czytniki i moduł perforatora.

Dążenie do maksymalnego skrócenia czasów reakcji na przerwania spowodowało podjęcie decyzji o powieleniu programu obsługi monitorów dla poszczególnych stacji. Podobnie różne stacje systemu mają własne programy obsługi czytników. Poza potrzebą optymalizacji, na rozwiązanie takie wpłynął fakt podłączenia czytnika stacji 2 do KPZ, co pociąga za sobą różnice w programie obsługi w porównaniu z czytnikiem stacji 1.

Natomiast perforatory są obsługiwane przez jeden program główny wykonany w wersji "czystej", tzn. umożliwiającej niezależne wykorzystywanie go na przemian przez różne stacje systemu. Program ten nie jest chroniony przed przerwaniami. Pracuje na różnych polach dla poszczególnych stacji i jest inicjowany przez krótkie programy chronione przed przerwaniami, które obsługują poszczególne stacje. Takie rozwiązanie pozwoliło zaoszczędzić znaczny obszar pamięci.

5. WNIOSKI

Na podstawie przytoczonego opisu systemu MACS można wyciągnąć następujące ogólne wnioski:

1. ograniczenia narzucone przez sprzęt mają istotny wpływ na strukturę systemu operacyjnego systemu liczącego, który realizowany jest dla danego sprzętu. Z jednej strony są to rozszerzenia programowe pewnych funkcji sprzętu (np. programowe zapełnianie bufora monitora), z drugiej strony - dostosowanie struktury oprogramowania do wymagań sprzętu ograniczających swobodę organizacji współdziałania programów (wyższy priorytet programów obsługi urządzeń niż programów nadzorczych),
2. pewne funkcje systemu operacyjnego mogą być przejmowane przez sprzęt. Przykładem jest nietypowe podłączenie monitorów w systemie MACS,

3. projektowanie i realizacja systemów operacyjnych wymagają zachowania ustalonej kolejności działań:
   1) sformułowanie zadań projektowanego systemu operacyjnego,
   2) ustalenie konfiguracji sprzętu, która byłaby zdolna realizować te zadania,
   3) weryfikacja zadań ze względu na ograniczenia narzucone przez sprzęt,
   4) opracowanie struktury systemu operacyjnego,
   5) programowanie, uruchamianie i sporządzanie dokumentacji.

Nieprzestrzeganie wymienionej kolejności działań prowadzi do nadmiernych nakładów pracy.

4. wzrastająca ciągle złożoność systemów operacyjnych wskazuje na potrzebę opracowania łatwych w użyciu metod formalnego opisu budowy, wzajemnych powiązań i zachowania poszczególnych elementów systemu.

<u>Literatura</u>

[1] KARCZEWSKI J. i in.: Podręcznik użytkownika wielodostępnego systemu konwersacyjnego MACS, Prace COPAN (w druku),

[2] KARCZEWSKI J., RUSZKOWSKA A.: Podręcznik operatora systemu MACS, Prace COPAN (w druku),

[3] MAROŃSKA B.: Język konwersacyjny MARO, PWN, 1973.

СИСТЕМА С РАЗДЕЛЕНИЕМ ВРЕМЕНИ ТИПА "ВОПРОС-ОТВЕТ" MACS ЧАСТЬ II.ОБСЛУЖИВАНИЕ ВВОДА-ВЫВОДА

Резюме

В труде описывается функционирование и структура системы, которая была создана Вычислительным центром ПАН для ЭВМ ODRA-1204 .

MACS может использоваться в центрах, в которых работают ЭВМ ODRA-1204 с двумя или большим количеством запоминающих устройств на магнитном барабане с четырьмя станциями, для которых основным оборудованием является монитор, а внешним оборудованием перфоратор и считывающее устройство с перфорированной лентой.

При станциях системы одновременно могут работать четыре пользователя, а во вспомогательной памяти можно хранить 24 программы для более позднего использования. Система идентифицирует пользователя с помощью номера заказа (с I - 499) и личного, секретного пароля пользователя. Одновременно ведется бухгалтерский учет. В заключение сеанса каждый пользователь получает информацию о начислении расходов, обременяющих его заказ. Для оценки эффективности вводится статистическая оценка некоторых элементов системы. Статистические данные оператор может получать в заключение расчетного периода. Оператор должен вводить в систему и выводить информацию о работающем оборудовании, состоянии системы, заказах пользователей и т.п.

В описании структуры системы особое внимание обращено на влияние ограничений оборудования на строение системы.

A MULTIACCESS CONVERSATIONAL SYSTEM MACS
Part II. The Input-Output Organization

Summary

This paper presents a functional description and structure of the MACS system developed in Computation Centre of the PAS for ODRA 1204 computer.

MACS can be used in computation centres equipped with ODRA 1204 computers with two or more drum memory units and four terminals with a monitor as the basic equipment, and with paper tape punch and reader for input/output as the optional equipment.

Four users may work at those terminals simultaneously, and 24 programs may be stored for further use in the backing store. The job number (1 to 499) and a private, secret word are to identify the user. Charging for time and accounting are automatically conducted at the same time. At the end of conversation each of the users is obtaining the information on charges for his job. In order to enable the estimation of the system efficiency, statistical measures for some elements of the system were introduced. Statistical data can be outputted by the operator at the end of accounting period.

The operator's task is also to input and output the information on working hardware, system status, user's job, etc.

In the description of system's structure a particular attention is paid to the influence of hardware limits on the construction of the system.

ALGORYTMY, Zeszyt Specjalny, 1974, 233-239

681.322.06;
061.3/438/"1973"

O PEWNEJ REALIZACJI SYSTEMU WIELODOSTĘPNEGO NA EMC ODRA 1204

Janusz W. SOWIŃSKI
Wojskowy Instytut Łączności

Pracę złożono 10.I.1974

Podano opis realizacji modelu systemu wielodostępnego na EMC ODRA 1204 współpracującej z urządzeniem zobrazowania graficznego typu grafoskop. System umożliwia niezależną pracę na dwóch stanowiskach z możliwością prostej rozbudowy przez dołączenie nowych stanowisk podobnego typu. System napisany został w języku JAS-O i uruchomiony na EMC ODRA 1204.

Spis treści

1. WSTĘP

Artykuł dotyczy projektu i konstrukcji wyspecjalizowanego systemu wielodostępnego dla maszyny ODRA 1204 wyposażonego w monitory dalekopisowe (MD) i urządzenia zobrazowania graficznego tzw. monitory ekranowe (ME). ODRA 1204 nie była projektowana z myślą o wielodostępnej eksploatacji, tym niemniej

istnieją w niej środki układowe, ułatwiające projektowanie systemu wielodostępnego.

2. KONFIGURACJA SPRZĘTU

Dokładną charakterystykę sprzętu zawiera dokumentacja techniczno-ruchowa EMC ODRA 1204 oraz opis techniczny lokalnego urządzenia zobrazowania graficznego LUZ.

W omawianym systemie wykorzystano następujące urządzenia:

- jednostkę centralną (JC z koordynatorem kanałów i dwoma kanałami transmisji danych)
- elektryczną maszynę do pisania (EMP)
- lokalne urządzenie zobrazowania graficznego (LUZ)
- pamięć operacyjną o pojemności 48K, przy czym bezpośrednio może być adresowanych 16K słów

Natomiast bazę językową systemu stanowi język niskiego poziomu JAS-0, którego opis zawiera opracowanie WZE ELWRO "System Programowania. Maszyna cyfrowa ODRA 1204. Język Adresów Symbolicznych".

3. FUNKCJE SYSTEMU

W omawianym systemie z maszyny korzysta się za pomocą języka konwersacyjnego jednolitego dla obu typów urządzeń zewnętrznych.

Konwersacja odbywa się w następujący sposób. Użytkownik "pisze" na ekranie (lub dalekopisie) pewne wyrażenie, nadając zmiennym wartości i sygnalizując w odpowiednim momencie fakt zakończenia tej czynności. Odpowiedzią systemu jest wypisanie wartości wprowadzonego wyrażenia dla zadanych parametrów.

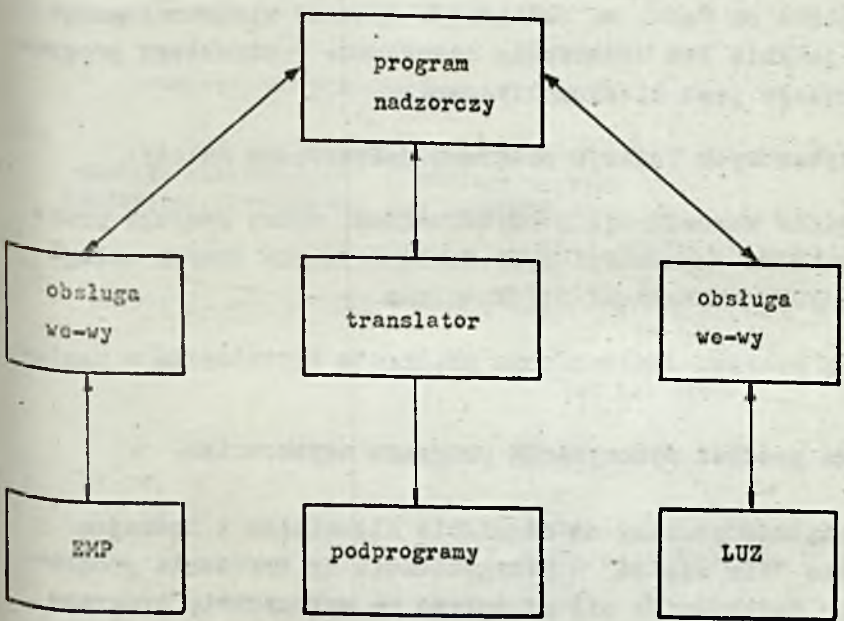
System reaguje na bieżąco na błędy użytkownika, sygnalizując jednocześnie ich charakter, dzięki temu użytkownik unika pisania do końca wyrażenia, w którym znajduje się błąd.

Ponieważ dostępne były jedynie dwa urządzenia zewnętrzne typu konwersacyjnego tzn. LUZ i EMP, realizacja systemu obejmuje jedynie te dwa urządzenia, czyniąc system dostępny w praktyce dwóm użytkownikom.

4. STRUKTURA SYSTEMU

4.1. Schemat strukturalny systemu

Schemat strukturalny oprogramowania ilustruje rys. 1.



Rys. 1. Schemat strukturalny oprogramowania

Urządzenia zewnętrzne kontaktują się z programem nadzorczym (i odwrotnie) przez programy obsługi urządzeń we-wy uwzględniające specyfikę stacji.

Translator języka konwersacyjnego pracując pod kontrolą programu nadzorczego jest wykorzystywany jednocześnie przez obu użytkowników systemu. W pamięci przechowywana jest jedna

kopia translatora. Każdy z użytkowników dysponuje obszarem roboczym, który jest wykorzystywany podczas pracy translatora.

4.2. Program nadzorczy

Program nadzorczy jest elementem oprogramowania podstawowego odpowiedzialnym za wykorzystanie zasobów systemu i obsługę żądań użytkowników w sposób zgodny z założeniem systemu, tzn. w sposób wielodostępny.

Ze względu na fakt, że realizacja systemu wielodostępnego obejmuje jedynie dwa urządzenia zewnętrzne - struktura programu nadzorczego jest nieskomplikowana.

Do podstawowych funkcji programu nadzorczego należy:

1) utrzymanie komunikacji z użytkownikami przez obsługę przerwań kanałów transmisji oraz zabezpieczenie innych usług związanych z konsolami użytkowników

2) obsługa procesu umieszczania programów użytkownika w pamięci

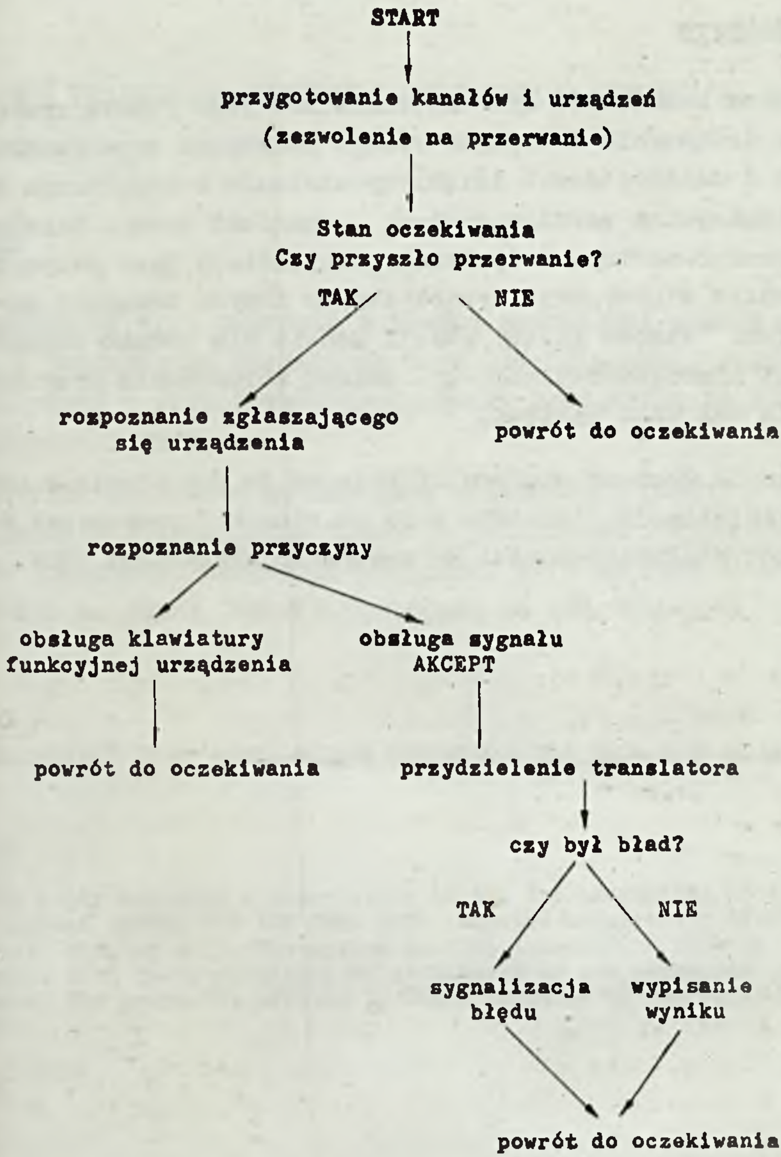
3) obsługa procesu wykonywania programu użytkownika.

Dwa ostatnie procesy są od siebie niezależne i nawzajem nic o sobie "nie wiedzą" - przygotowanie do wykonania programu jednego użytkownika nie ma wpływu na wykonywanie programu drugiego użytkownika (jeżeli nie brać pod uwagę, że czas wykonywania wydłuża się).

Każdy proces zawiera instrukcje odwołujące się do fragmentu programu nadzorczego, który inicjuje działanie kanałów.

Ze względu na ograniczoną (np. rozmiarem ekranu LUZ) wielkość formuł arytmetycznych jakie mogą pisać użytkownicy, nie zachodzi potrzeba dynamicznego przydziału pamięci.

Strukturę programu nadzorczego przedstawia rys. 2.



Rys. 2. Struktura programu nadzorczego

5. ZAKOŃCZENIE

Omówiona realizacja systemu umożliwia jego prostą rozbudowę przez dołączenie kolejnych stacji podobnego typu (monitory ekranowe i dalekopisowe) dzięki wydzieleniu w strukturze systemu odpowiednich modułów obsługi urządzeń we-wy. Doświadczenia przeprowadzone z systemem potwierdzają jego przydatność również w przypadku wykorzystania innych urządzeń zewnętrznych. Wzrost liczby stacji będzie się jednak wiązał z pewnymi niedogodnościami, np. zmianą adresowania przy przekroczeniu 16K słów pamięci.

Wykonanie systemu nie kwalifikuje go do powielania i użytkowej eksploatacji, jednakże może on stanowić przyczynek do realizacji wielodostępności na maszynach klasy ODRA 1204.

ОБ ОДНОЙ РЕАЛИЗАЦИИ СИСТЕМЫ С РАЗДЕЛЕНИЕМ ВРЕМЕНИ НА БАЗЕ ЭВМ ODRA-1204

Резюме

Настоящий труд содержит описание реализации модели системы с разделением времени на базе ЭВМ ODRA-1204, взаимодействующей с устройством отображения информации на ЭЛТ графического типа.

Система позволяет производить независимую работу с двумя станциями с возможностью простого расширения оборудования путем добавления очередных устройств того же типа. Система написана на языке JAS-0 и проверена на ЭВМ ODRA-1204.

AN IMPLEMENTATION OF MULTI-ACCESS SYSTEM FOR THE ODRA 1204 COMPUTER

Summary

This paper contains a description of the implementation of the Multi-Access system for the ODRA 1204 computer cooperating with the graphical display unit. The system enables independent work on two terminals with the possibility of attachment of new terminals of similar type. The system is written in JAS-0 language on the ODRA 1204 computer.

ALGORYTMY, Zeszyt Specjalny, 1974, 241-255

681.322.06:
061.3/438/"1973"

SYSTEM OPERACYJNY EMC KAR-65

Andrzej GECOW
Instytut Badań Jądrowych
Pracę złożono 10.I.1974

Kar-65 jest małą doświadczalną maszyną cyfrową trzeciej generacji. Jej głównym zadaniem jest współpraca z dwoma aparatami pomiarowymi. Opracowanie dotyczy budowy systemu operacyjnego i jego głównej części - dyrygenta, ściśle dostosowanych do specyficznych potrzeb instalacji. Pokazano sposób realizacji ekstrakodów typu "reentrant" oraz podłączenia obsługi aparatów pomiarowych przy podziale czasu. Ponadto pokazane są przykłady pewnych rozszerzeń systemu dokonywanych w "biegu".

Spis treści

1. HISTORIA I PRZEZNACZENIE MASZYNY

EMC Kar-65 jest małą doświadczalną maszyną cyfrową o logice trzeciej generacji opartej na przerwaniach.

Zbudowana została w 1965 r. w Instytucie Fizyki Doświadczalnej UW.

Głównym przeznaczeniem maszyny jest opracowywanie danych z komór pęcherzykowych, polegające na rekonstrukcji geometrycznej i dynamicznej zderzeń cząstek elementarnych oraz statystycznym opracowaniu wyników. Rocznie mierzono kilkadziesiąt tysięcy zdjęć, a opracowywanie ich na maszynie Gier zajęłoby około pięciu godzin dziennie. W celu zwiększenia wydajności aparatów pomiarowych podłączono je bezpośrednio do maszyny.

Drugim głównym zajęciem maszyny jest liczenie długo działających programów do badań naukowych, zawierających zazwyczaj całki wielokrotne. Programy takie dają stosunkowo bardzo krótkie wydruki przy bardzo długim (do kilkudziesięciu godzin) czasie liczenia.

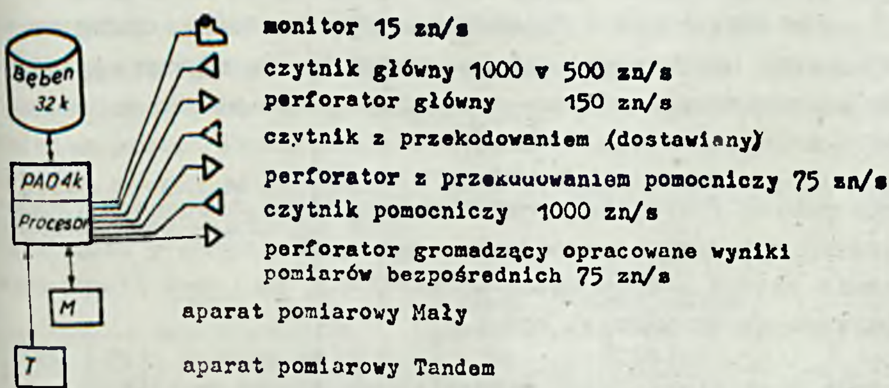
Bezpośredni dostęp do maszyny w Instytucie pozwalał także na konwersacyjną pracę z kilkoma usługowymi programami typu Edit - do poprawiania taśmy papierowej lub Arytmometr - do konwersacyjnego obliczania wartości funkcji i całek.

2. WŁAŚCIWOŚCI SPRZĘTU I JEGO KONFIGURACJA

Kar-65 ma 4K 26-bitowych słów pamięci operacyjnej (PAO) i 32K pamięci bębnowej. Rozkazy są jednoadresowe i mieszczą się w pojedynczym słowie maszyny. Konfiguracja maszyny pokazana jest na rys. 1.

Praca maszyny może odbywać się w dwóch stanach: legalności i nielegalności, nazywanych inaczej stanami dyrygenta i użytkownika. W stanie użytkownika uwzględniane są elektroniczne blokady PAO, zabezpieczające system i inne programy przed zniszczeniem przez program użytkownika. W tym stanie nieosią-

galna jest grupa rozkazów zwanych nielegalnymi, a używanych w stanie dyrygenta do organizacji wejścia-wyjścia. Użyć wejścia-wyjścia można przez ekstrakody, które przenoszą akcję do dyrygenta.



Rys. 1. Konfiguracja EMC Kar-65

Podstawową właściwością logiki trzeciej generacji są przerwania, pozwalające organizować podział czasu. Kar-65 ma 24 przerwania związane m.in. ze współpracą z aparatami pomiarowymi, oraz gotowością kanałów zewnętrznych itp. Każde przerwanie zostawia ślad w miejscu 0 i rozpoczyna akcję od określonego miejsca pamięci, wprowadzając maszynę w stan "nie wolno przerywać". Stan ten dotyczy wszystkich pozostałych przerwań i dopiero stan "wolno przerywać" pozwala na realizację przerwań czekających. Jeżeli czeka kilka przerwań, to wykonują się one kolejno wg wyznaczonego priorytetu.

3. ZADANIA SYSTEMU

W 1970 r. rozpoczęły się prace nad podłączeniem do maszyny dwóch aparatów pomiarowych o nazwach Mały i Tandem. Równocześnie należało podsumować doświadczenia w pracy z maszyną i systemem (głównie dyrygentem) oraz przygotować włączenie programów obsługi tych urządzeń do systemu.

Specyficzna sytuacja jednostkowej doświadczalnej maszyny, oprogramowanej w assemblerze przez grupkę kilku fizyków, spowodowała zaostrzenie wymagań w stosunku do weryfikacji założeń i struktury systemu: z jednej strony konieczność dalszego działania całego dotychczasowego oprogramowania, w dużej części istniejącego tylko w postaci taśm binarnych; z drugiej strony, ponieważ praca z tą maszyną jest w zasadzie konwersacyjna i każdy użytkownik jest równocześnie operatorem, a więc sposób korzystania z systemu operacyjnego nie powinien ulec większym zmianom.

Mała pamięć operacyjna przy częstych długich czasach liczenia dużych programów spowodowała dużą cenę miejsca w PAO. Równocześnie częste awarie bębna spowodowały, że przez długi czas nie używano go do ważnych zadań.

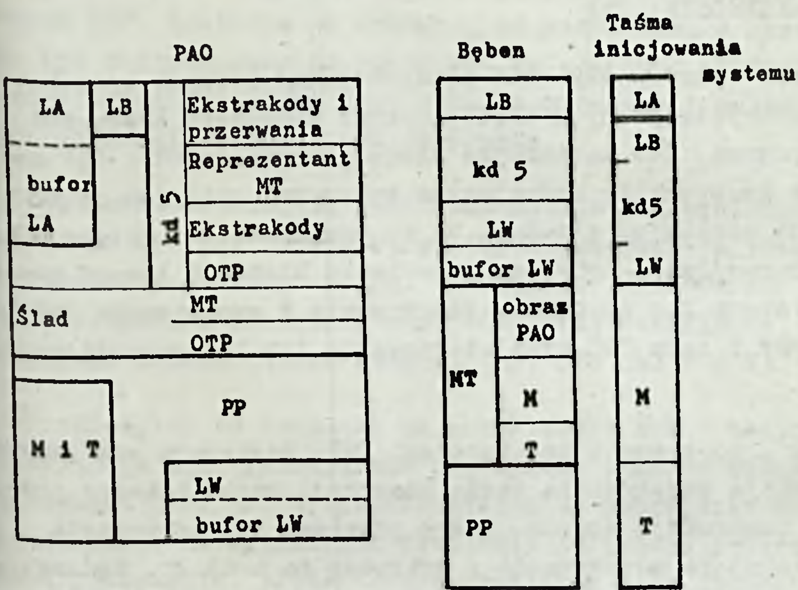
W tych dość nietypowych warunkach,przez budowę kolejno coraz bardziej zaawansowanych systemów i ich weryfikację w normalnej pracy maszyny powstawał nowy system zdolny do współpracy z urządzeniami przy podziale czasu.

Trzeba było pamiętać przy oddawaniu do użytku(a tym samym do sprawdzenia)nowego systemu, że rezygnowanie z nowych własności osiągniętych już przez ten system jest w zasadzie niemożliwe. Powstała w ten sposób seria systemów od skd1 (760 oktalnie miejsc PAO - super krótki dyrygent) do systemu dyrygenckiego sd1 z tzw. krótkim dyrygentem kd5. Górnym umownym ograniczeniem zajętości PAO przez system jest 2000 miejsc oktalnie.

4. SYSTEM DYRYGENCKI sd1

System dyrygencki sd1 powstał w maju 1972 r. i jest najbardziej rozwiniętym systemem. Obecnie już z mniejszym pośpiechem pracuje się nad następnym systemem. Statyczna budowa systemu przedstawiona jest na rys. 2. System wprowadzany jest do maszyny z taśmy papierowej. Najpierw wczytywany jest elektronicznie pomocniczy program o nazwie "Lokaj Anatol" (LA) znajduja-

cy się na początku taśmy, który następnie uruchamiany przez operatora, wczytuje dalszą część taśmy na początek bębna, gdzie zwykle znajduje się cały system. Skończywszy, LA przekazuje kontrolę "Lokajowi Bartkowi" (LB), który wchodzi na miejsce LA i po przygotowaniu obliczenia sumy kontrolnej wprowadza na miejsce, w którym się sam znajduje program - kd5. Kontrolę przekazuje mu przez przerwanie gotowości bębna. Przerwanie to za pierwszym przebiegiem realizuje się odmiennie. Liczy się wtedy suma kontrolna i przestawia program przerwania gotowości bębna na właściwą pracę. Można również wywołać "Lokaja Bartka" za pomocą klucza na pulpicie. Każdy z tych programów melduje na monitorze prawidłowe wykonanie się.



Rys. 2. Statyczna budowa sd1

W skład systemu wchodzą więc LA, LB i kd5. Rozszerzony system zawiera także programy obsługi urządzeń Mały i Tandem (M i T), wczytywane z taśm binarnych przez dyrygenta kd5. Obszar nie zajęty przez system wykorzystuje program programisty (PP). Zainicjowanie działania systemu jest odtwarzające w takim sensie, że PP może być kontynuowany, gdy awaria systemu nastąpiła w czasie wykonywania programów systemowych (OTP lub MT).

Krótki dyrygent (wersja 5) kd5 zawiera:

- obsługę kanałów wejścia-wyjścia przez ekstrakody i przerwania,
- obsługę błędów,
- system operacyjny OTP,
- reprezentanta obsługi MT wykonującego pierwsze i ostatnie czynności związane z obsługą przerwań Małego i Tandemu.

Ze względu na oszczędności pamięci, biblioteka programów użytkowych przechowywana jest na taśmach papierowych.

5. SYSTEM OPERACYJNY OTP

Do systemu operacyjnego OTP przejść można różnymi sposobami: przez wykrycie pewnych błędów, przez specjalny ekstrakod kończący program lub naciskając klucz przerwania OTP. OTP zawiera wiele instrukcji, które można wykonywać wpisując je z monitora lub wczytując z taśmy. Są to możliwości: ingerencji w pamięć operacyjną i bębnową, ustawiania blokad i innych parametrów systemu lub programu, wczytywania i wypisywania taśm binarnych "W" i taśm "R" oraz startowania lub kontynuacji programów.

Taśmy "R", to taśmy z instrukcjami OTP. Zawierają one zazwyczaj instrukcje wczytywania taśmy binarnej, taśmę binarną programu oraz instrukcję startu. Start programu lub instrukcja "o", przerzucające wczytywanie z powrotem na monitor, kończą taką taśmę. Rzadko używane duże programy realizacji instrukcji OTP, jak np. wczytywanie i wypisywanie taśm binarnych bębnowych oraz suma kontrolna bębna przeniesione zostały na bęben (LW) i nie zajmują stale miejsca w PAO. Awaryjność maszyny i konieczność analizy zaistniałych błędów w normalnych warunkach sugerowały, aby nie przenosić pozostałych instrukcji na bęben. W przypadku dłuższej awarii bębna, kd5 można wczytać z taśmy i będzie on w pełni sprawny przy pracach nie związanych z bębnem, zachowując standardy taśm "R" i kontynuacji programów.

OTP pozwala sięgnąć do każdego miejsca PAO lub bębna i dowolnie je zmieniać. Obszar systemu jest jednak chroniony i dopiero po podaniu hasła można go oglądać i zmieniać. Dotyczy to również wczytywania taśmy "R" z programami obsługi Małego i Tandemu na zarezerwowane pole bębna. Ochrona ta ma za zadanie ustrzec system przed pomyłkami operatorów.

6. WIELODOSTĘPNOŚĆ * EKSTRAKODÓW

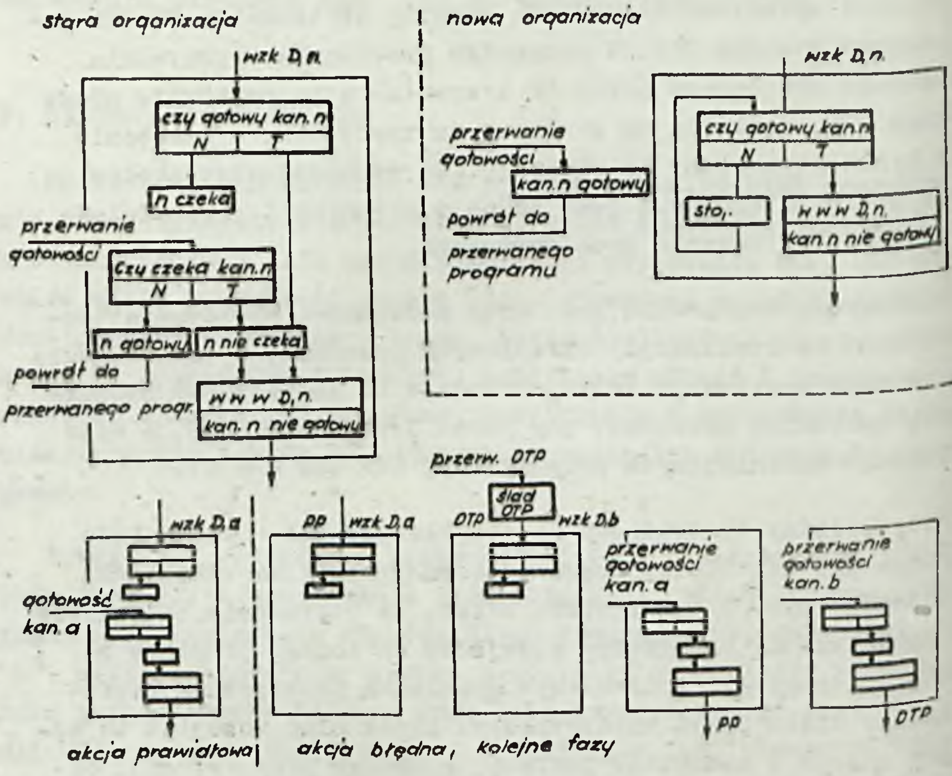
Program wykonywać można jako program użytkownika PP lub jako program systemu OTP. W przypadku przerwania wykonywania ekstrakodu wywołanego przez PP i wywołania go powtórnie przez program OTP, zostanie on wykonany na rzecz OTP, a następnie może być kontynuowany na rzecz PP. Ta własność ekstrakodów zwana wielodostępnością jest jedną z głównych i najtrudniejszych do zrealizowania cech dyrygenta.

Podstawową trudnością jest brak możliwości wybierania (dopuszczania do realizacji) określonych przerwań, a równocześnie wzbraniania realizacji innym. Powoduje to konieczność dopuszczania wszelkich przerwań, gdy jedno jest oczekiwane, a więc i przerwań zmieniających program (np. OTP lub M i T).

Prześledzimy tę trudność na przykładzie wzk - ekstrakodu wczytującego lub wypisującego pojedynczy znak na wskazanym urządzeniu (rys. 3). Z rysunku widać, że poprzednia budowa wzk nie pozwalała na bezpieczne przejście do innego programu za pomocą przerwania. Podstawowym problemem jest jednak zabezpieczenie śladu przed zniszczeniem. Przez ślad rozumieć tu będziemy obszar i zawartość pamięci, w którym przy wejściu do podprogramu maszyna zapamiętuje rejestry i parametry. Przy wychodzeniu z tego podprogramu ślad wkładany jest z powrotem, czyli rejestry i parametry są regenerowane do stanu sprzed wejścia do podprogramu.

* Przyp. redakcji: Wielodostępność jest tu rozumiana jako możność ponownego wejścia do tego samego programu, zanim wykona się on do końca. W literaturze anglosaskiej używa się w tym znaczeniu słowa "reentrance".

Ślad zostawiany przy wejściu do wzk jest znikomy i ogranicza się do śladu pozostawionego elektronicznie zawsze przy wejściu do każdego ekstrakodu. Pozostałe ekstrakody często wywołują wzk ze swojego wnętrza. Są one zazwyczaj bardziej skomplikowane i potrzebują kilku (nie wszystkich) rejestrów. Ponieważ jeden program może być w jednym czasie tylko w jednym ekstrakodzie, wystarczy jeden wspólny podprogram pozostawiania śladu i regeneracji rejestrów.



Rys. 3. Stara i nowa organizacja wzk

Zawsze przy przejściu do OTP z PP ślad ten przenoszony jest na inny obszar. Ślad pozostawiony elektronicznie oraz komórki robocze ekstrakodów porozrzucane są po pamięci i nie stanowią zwartego bloku. Te miejsca także zostają przeniesione. Teraz dopiero OTP może korzystać z ekstrakodów. Przed powyższymi ope-

racjami, na samym początku OTP zapamiętuje stan rejestrów. W ten sposób powstaje ślad OTP, w którym znajduje się zawartość rejestrów i stan dyrygenta właściwego (tzn. wielodostępnych ekstrakodów).

Umiejscowienie śladu OTP za kd5 w PAO, a tuż przed PP, pozwala na przeniesienie śladu wraz z PP w jednym bloku na taśmę binarną lub bęben, i nie narażając chronionego obszaru systemu pozwala wprowadzić z powrotem na to samo miejsce. W międzyczasie można zniszczyć ślad w PAO, tzn. np. liczyć inne programy. Powyższa organizacja umożliwia przerwanie i wznowienie w dowolnym miejscu każdego programu.

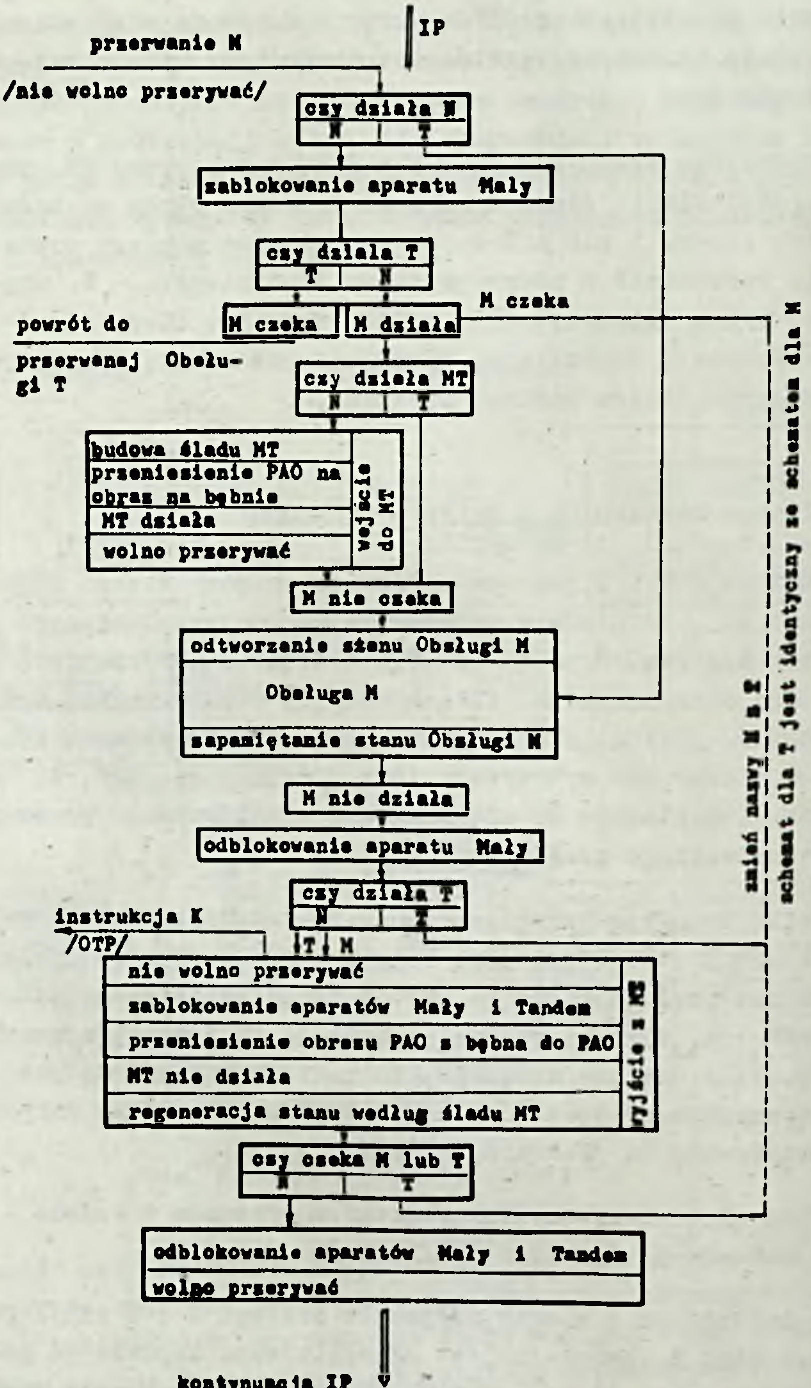
7. ORGANIZACJA WSPÓŁPRACY Z MAŁYM I TANDEMEM

Ślad przerwań M i T jest umiejscowiony między śladem OTP a programem OTP, co ułatwia zachowanie go podczas ponownego inicjowania systemu. Ślad MT zawiera dodatkowo informacje o przerwaniach operatorskich, takich np. jak OTP. W czasie działania programu obsługi MT, wszystkie przerwania operatorskie, po których miałby być wykonywany inny program np. OTP, są ignorowane. Organizację MT opracowaną i zrealizowaną przez Marka Szczekowskiego przedstawia rys. 4.

Na bębnie istnieją dwa niezależne, równoprawne i różne programy obsługi M i T. Różne jest samo wnętrze programów obsługi, natomiast przedstawiona na rys. 4 organizacja zewnętrzna jest identyczna. Gdy przerwanie pochodzące od jednego aparatu nastąpi podczas obsługi drugiego, to zostaje ono zapamiętane, i dalej kontynuowany jest przerwany program, a po jego zakończeniu przerwanie to jest symulowane.

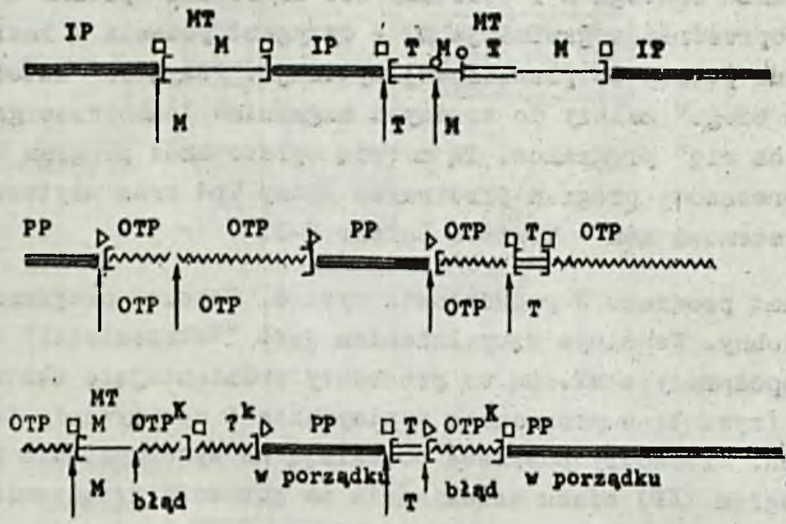
Przebieg współpracy obydwu urządzeń z systemem w czasie pokazuje schemat na rys. 5.

Pełna informacja o stanie programów obsługi M i T znajduje się wraz z nimi na bębnie i jest aktualizowana dopiero po pomyślnym opracowaniu porcji danych. Pozwala to na ponowne wpro-



Rys. 4. Schemat współpracy systemu z programami obsługi aparatów pomiarowych

wadzenie i inicjowanie dyrygenta w każdej chwili, bez straty uzyskanych pomiarów. Można także na nowo wprowadzić programy MT i zainicjować je nie niszcząc PP, a jedynie przerywając go na chwilę.



[udostępnienie ekstrakodów /i PAO dla MT/

] regeneracja ekstrakodów /i PAO dla MT/

○□▷ ślady

Rys. 5. Przykładowe przebiegi czasowe wymiany programów

8. "OPCJE W BIEGU" *

Pomiary na aparatach M i T wykonywane są bez przerwy, a więc nigdy nie ma takiego czasu maszyny, w którym można by było bez szkody dla pomiarów robić doświadczenia z nowymi rozwiązaniami dyrygenta. Wstrzymywanie pomiarów jest niepożądane. Głównym problemem, który ma być rozwiązany w następnym systemie jest buforowanie wejścia-wyjścia. W doświadczeniach trzeba więc manipulować programami reakcji na przerwania.

* Przyp. redakcji: Pojęcie "opcja" oznacza tu pewną dodatkową funkcję systemu operacyjnego uzyskiwaną przez jego rozszerzenie. "Opcja w biegu" oznacza rozszerzanie systemu w czasie jego pracy.

W tej sytuacji przyjęta została metoda "opcji w biegu" polegająca na reorganizacji systemu przez interpretację taśmy "R". Dyrygent zmieniany jest podczas własnej pracy i często się zdarza, że końcowe instrukcje OTP mają już nową treść. Równocześnie obsługa M i T działa bez zakłóceń. Opisana w rozdziale poprzednim organizacja MT - dyrygent pozwala w razie pomyłki na powrót do prawidłowej sytuacji. Powyższa metoda "opcji w biegu" należy do trudnych zagadnień "samoprzeorganizowujących się" programów. Tą metodą opracowano: program W i V, trójprocesowy program przetwarza taśmy Tb1 oraz użytkową opcję systemową mb2 - Monitor Buffer V-2.

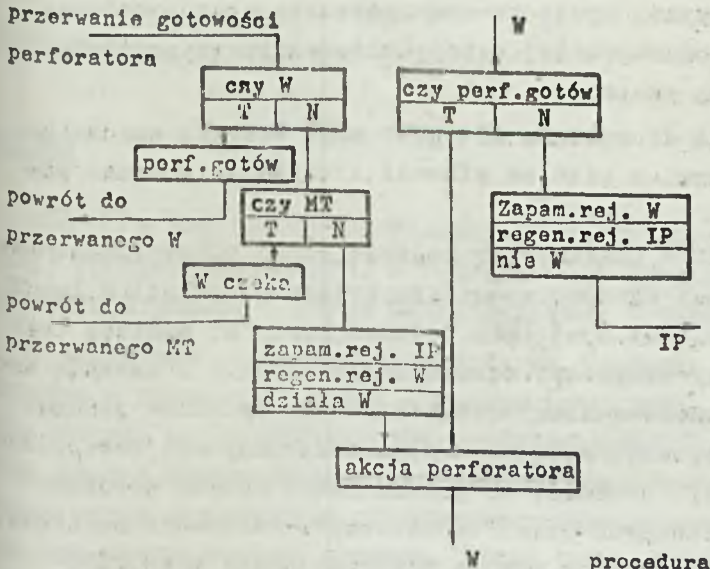
Schemat programu W przedstawia rys. 6. Schemat programu V jest podobny. Wspólnym uzupełnieniem jest "Wskrzesiciel" dodany do współpracy z MT. Są to procedury podmieniające ekstrakod wzk (rys. 3) w programach wypisywania i wczytywania taśm binarnych. Procedury powyższe pozwalają na wykorzystanie przez inny program (IP) czasu oczekiwania na gotowość urządzenia. Jednak gdy działa program obsługi MT, przerwanie z gotowego urządzenia nie może być obsłużone, gdyż obszar, którego dotyczy obraz taśmy binarnej w PAO przeniesiony został na bęben. O nie obsłużonych przerwaniach "przypomina sobie" periodycznie program "Wskrzesiciel" (wykorzystując przerwania zegarowe).

W opcji Tb1 rejestry zostały przypisane do procesów bądź podzielone między nie - według określonych reguł. Pozwoliło to uniknąć czasochłonnego ich zachowywania i regeneracji. Opcja ta składa się z następujących trzech procesów (pięciu elementów):

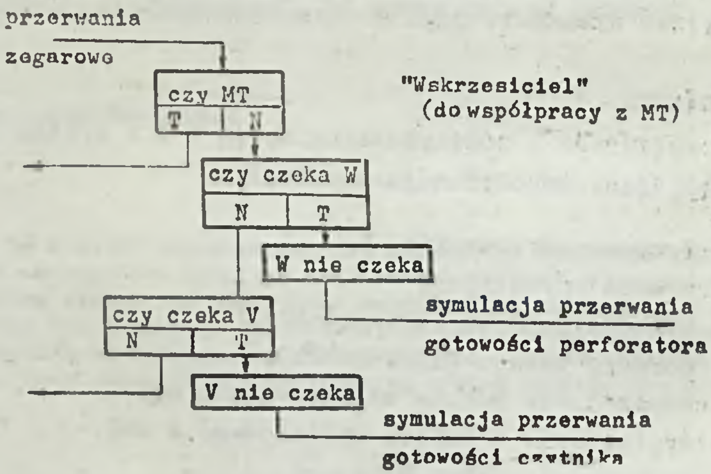
1) wczytania znaku z taśmy do bufora wejściowego,
2) a) wyjęcia znaku z bufora wejściowego,
   b) opracowania go przez program,
   c) wpisania wynikowego tekstu do bufora wyjściowego,

3) wypisania znaku z bufora wyjściowego.

Ponieważ programy procesów 1 i 3 działają w stanie "nie wolno przerywać", przerwanie OTP może nastąpić jedynie w czasie realizacji procesu 2.



procedura zastępująca wzk dla opcji W



"Wskrzesiciel" (do współpracy z MT)

Rys. 6. Schemat procedury W i programu "Wskrzesiciel"

Zastosowana tu metoda pozwala znacznie przyśpieszyć realizację programów, które mogą podporządkować się narzuconym ograniczeniom, a limitowane są czasem wczytania lub wypisania taśmy. Programy procesów 1 i 3 oraz bufory znajdują się na obszarze PAO nie przenoszonym przez MT. Mogą one działać także podczas pracy programów obsługi M i T, gdyż wykorzystują tylko jeden rejestr, którego regeneracja nie zabiera zbyt dużo czasu.

Dwie omówione powyżej opcje to doświadczenia przeprowadzone w celu dobrania odpowiedniej metody buforowania z podziałem czasu w przyszłym systemie.

Użytkową opcją do systemu sd1 jest mb2. Pozwala ona na buforowanie monitora, a więc na pisanie i czytanie podczas pracy MT.

Obsługa aparatów pomiarowych zajmuje ponad połowę czasu pracy maszyny, a czas każdego wywołania programu obsługi M lub T wynosi około 3 s. Bez opcji mb2 podczas pracy MT monitor był zablokowany. Przy zazwyczaj konwersacyjnej pracy z maszyną bardzo częste 3-sekundowe czasy oczekiwania na wpisanie jednego znaku okazały się zbyt męczące. Opcja mb2 zbudowana jest podobnie jak opcja Tb1. Dochodzi tu jednak kilka nowych warunków współpracy wyznaczonych przez techniczne właściwości monitora. Program, który w praktyce wraz z buforami zajął tylko 200 miejsc oktalnie PAO, osiągnął tak wielkie skomplikowanie współzależności, że wykazanie jego poprawności wydawało się nieosiągalne.

9. PODSUMOWANIE

Wraz z opcją mb2 i doświadczalną opcją W i V system sd1 zawiera następujące współpracujące procesy:

1) PP - program użytkownika,
2) OTP - system operacyjny,
3) T - obsługa Tandemu (element MT),
4) M - obsługa Małego (element MT),
5) pka - napełnianie bufora wejściowego w mb2,
6) pkb - wypisywanie z bufora wyjściowego w mb2,
7) W - wypisywanie taśmy binarnej,
8) V - wczytywanie taśmy binarnej,
9) "Wskrzesiciel" - pomocniczy program dla opcji W i V.

Korzystając ze standardowych taśm "R" zawierających zespół instrukcji przeniesienia i kontynuacji programów, można powiększyć dowolnie liczbę PP.

Jedynie wielkość PAO potrzebna dla MT podczas wykonywania obsługi Małego i Tandemu uniemożliwia równoczesne uruchomienie dwóch dodatkowych procesów opcji Tb1.

ОПЕРАЦИОННАЯ СИСТЕМА ЭВМ KAR-65

Резюме

Kar-65 - это экспериментальная вычислительная машина (ЭВМ) третьего поколения. В основном она предназначена для совместной работы с двумя измерительными установками.

В настоящей разработке обсуждается строение операционной системы и главная ее часть - программа-дирижер, тесно согласованные со специфическими требованиями ЭВМ. Представляется способ осуществления экстракодов типа "reentrant" , а также метод подключения обслуживания измерительных установок при распределении времени. Кроме того, приводятся примеры добавочных возможностей расширения системы с разделением времени, производимого во время работы системы.

THE KAR-65 OPERATING SYSTEM

Summary

Kar-65 is a small experimental 3rd generation computer. Its main task is on-line cooperation with two measurement devices. Discussion concerns the operating system and supervisor construction which suit specific installation requirements. Implementation of reentrant extracodes and of software interface between the system and the devices serviced on-line are shown. Some examples of time-sharing system options are given.

ALGORYTMY, Zeszyt Specjalny, 1974, 257-266

681.322.06:
061.3/438/"1973"

TELSPIS - SYSTEM GROMADZENIA I WYSZUKIWANIA INFORMACJI O ABONENTACH TELEFONICZNYCH

Wiesław BABCZENKO

Okręgowe Laboratorium
Poczty i Telekomunikacji

Pracę złożono 10.I.1974

W opracowaniu przedstawiono system gromadzenia i wyszukiwania informacji "TELSPIS" wykonywany przez Ośrodek Informatyki Technicznej i Przetwarzania Danych w OLPiT w Warszawie w ramach systemu "TELSIT". Omówiono zagadnienie modularności systemu. Przedstawiono koncepcję pamięci hierarchicznej i zarządzania danymi w takiej pamięci.

S p i s t r e ś c i

1. CHARAKTERYSTYKA SYSTEMU TELSPIS

TELSPIS wchodzi w skład systemu TELSIT, który jest systemem automatyzującym podstawowe usługi telekomunikacyjne dla ludności i składa się z następujących podsystemów: TELSART, TELLOK, TELKUR, TELSERWIS, TELSPIS.

TELSART jest systemem rozliczeń telefonicznych, tzn. świadczy usługi w zakresie rejestracji liczby rozmów dla poszczególnych abonentów, wystawiania rachunków telefonicznych i wykonywania różnego rodzaju zestawień i statystyk.

TELLOK jest systemem informacji lokalnej o dyżurach aptek i szpitali, repertuarze kin i teatrów, muzeach, wystawach, itp.

TELKUR świadczy usługi w zakresie biura zleceń, tzn. budzi i przypomina o różnych terminowych sprawach.

TELSERWIS świadczy usługi w zakresie biura napraw, a więc prowadzi ewidencję uszkodzeń i napraw oraz związanych z nimi rozliczeń.

TELSPIS jest systemem gromadzenia i wyszukiwania informacji realizującym funkcje biura numerów, tzn. gromadzi dane o abonentach telefonicznych, aktualizuje posiadaną informację, udziela na żądanie informacji o abonentach, wydaje różnego rodzaju spisy i katalogi.

System TELSPIS charakteryzuje się następującymi cechami:

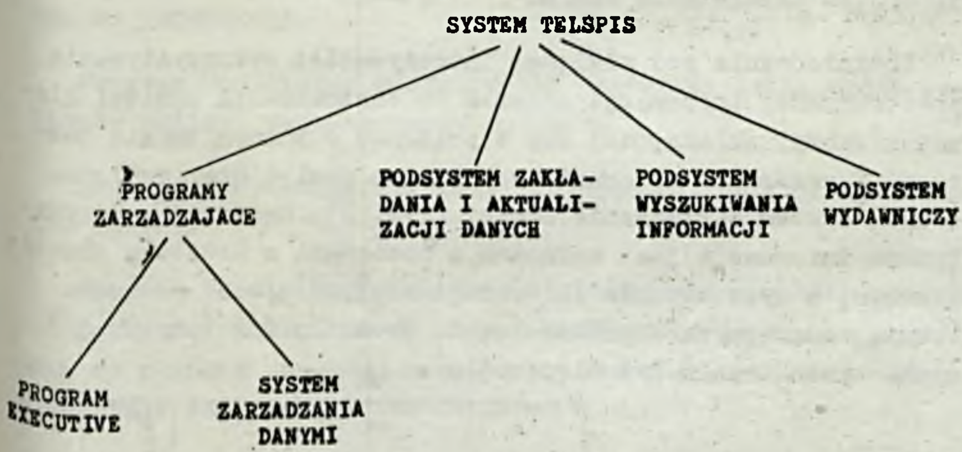
- możliwością przechowywania 200 tys. ÷ 1 mln dokumentów źródłowych,
- możliwością przyjmowania 300 ÷ 600 dokumentów dziennie. Dokumenty te na ogół dotyczą zmian w istniejących już zapisach,
- liczbą usług: 30 ÷ 50 tys. dziennie,
- wielodostępnością : jednoczesna obsługa 16 ÷ 32 stacji niezależnych użytkowników,
- średnim czasem reakcji: 5 ÷ 15 s,
- modularną konstrukcją umożliwiającą modyfikację i rozbudowę systemu.

2. MODULARNOŚĆ SYSTEMU

Aby poznać i opanować dowolny skomplikowany proces staramy się go uprościć wyróżniając w nim moduły (bloki funkcjonalne) realizujące prostsze procesy składowe i określając powiązania między nimi. Takie postępowanie można kontynuować dzieląc duże moduły na mniejsze.

Postępując tak przy projektowaniu systemów oprogramowania otrzymamy zbiór modułów, tzn. funkcjonalnie wyodrębnionych sekwencji rozkazów, które kontaktują się ze sobą w standardowy sposób.

Takie podejście zastosowano przy projektowaniu systemu TELSPIS. Składa się on z modułów podsystemów (rys. 1), z których każdy ma także budowę modularną. Przedstawienie systemu w postaci połączonych bloków funkcjonalnych sprawia, że jest on przejrzysty, co pozwala szybciej wychwycić błędy i nieprawidłowości.



Rys. 1. Struktura systemu TELSPIS

Porównanie struktur podsystemów umożliwiło wykrycie podobieństw, szczególnie w modułach przetwarzających zbiory danych, uogólnienie funkcji i unifikację modułów. Zmniejszenie

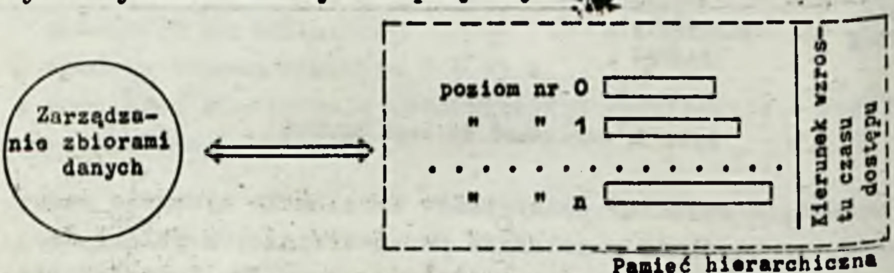
liczby różnych modułów przyczyniło się do obniżenia nakładu pracy przy realizacji systemu, a dokumentacja zyskała na przejrzystości.

Dołączając nowe moduły można sukcesywnie wdrażać system przechodząc od modelu do systemu pełnego. Stosunkowo łatwa wymiana modułów stwarza możliwość praktycznej oceny przydatności różnych algorytmów oraz dostosowania struktury systemu do zmieniających się wymagań użytkownika.

Uniwersalność i nadmierna liczba modułów wydłużają czas reakcji systemu, co może wywoływać opory realizatorów. Jednak możliwość przyspieszenia realizacji systemu i zwiększenia jego niezawodności na drodze wykorzystania już istniejących i sprawdzonych modułów świadczy na korzyść modularności, zwłaszcza przy zwiększającym się niedoborze odpowiedniej kadry pracowników.

3. SYSTEM ZARZĄDZANIA DANYMI

Zróżnicowanie pod względem intensywności wykorzystywania przetwarzanej informacji skłania do zastosowania pamięci hierarchicznej, składającej się z poziomów o różnym czasie dostępu. Zarządzanie zbiorami danych oraz pamięć hierarchiczna tworzą system zarządzania danymi (rys. 2). Częściej wykorzystywana informacja jest związana z poziomami o krótszym czasie dostępu, a wyszukiwanie informacji zaczyna się od poziomów łatwiej dostępnych i przechodzi do trudniej dostępnych, gdy wynik wyszukiwania był niepomyślny.



Rys. 2. System zarządzania danymi

Takie rozwiązanie pozwala osiągnąć odpowiednie parametry czasowe przy umiarkowanym wzroście kosztów.

W systemie TELSPIS stosowane są pamięci masowe o dostępie sekwencyjnym (taśmy magnetyczne) i bezpośrednim (dyski magnetyczne wymienne i stałe). Najmniejszą jednostką informacji, którą można adresować, jest blok, który ma jednakową długość we wszystkich typach pamięci.

W pamięci hierarchicznej wyróżniono cztery poziomy (pamięć operacyjna, dyski stałe, dyski wymienne, taśma magnetyczna), przy czym nie wszystkie poziomy muszą istnieć.

System zarządzania danymi wykorzystuje: program Executive, program realizacji poziomów PRP oraz program operowania zbiorami POZ (rys. 3), i daje użytkownikowi dwie możliwości: operowanie na spójnych zbiorach danych niezależnie od ich struktury i położenia oraz operowanie bezpośrednio na danych umieszczonych w poszczególnych poziomach pamięci (rys. 4). W obu przypadkach możliwy jest bezpośredni i sekwencyjny dostęp do informacji.

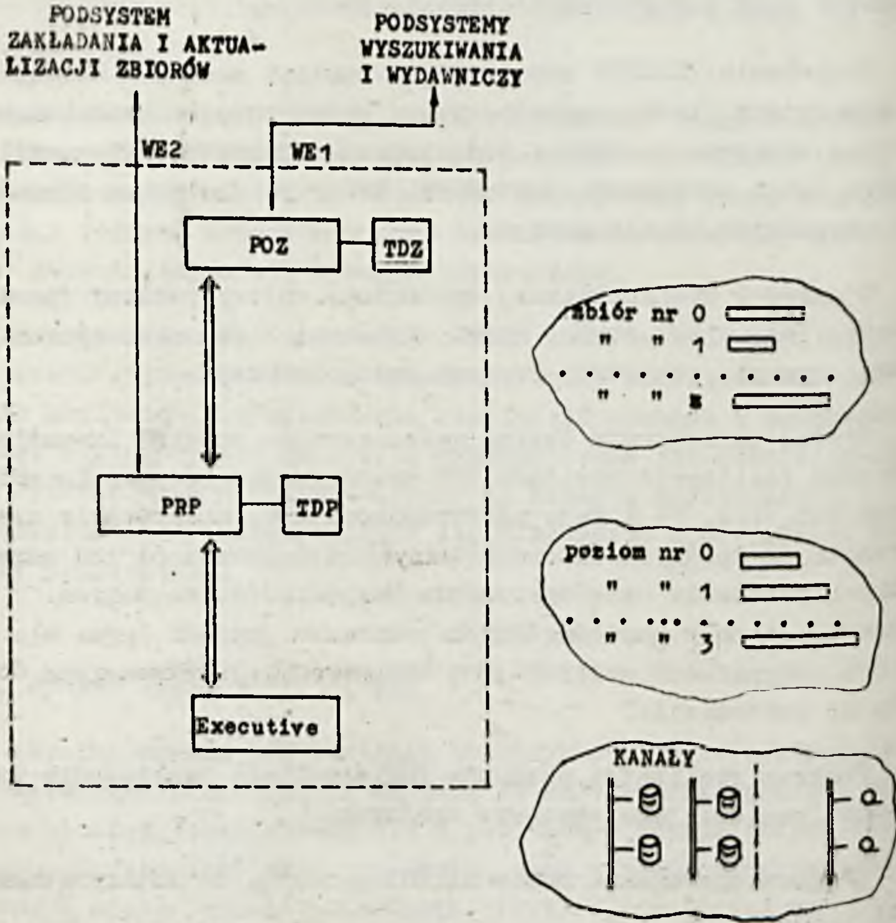
Program realizacji poziomów PRP umożliwia traktowanie poziomów pamięci jako obszarów spójnych.

Program operowania zbiorami POZ sprawia, że zbiory stają się spójne.

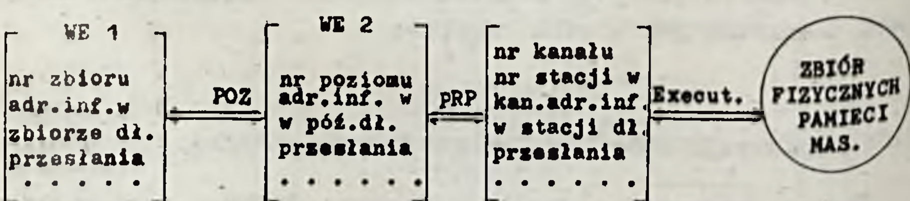
Program Executive (poza innymi funkcjami) zarządza zbiorem fizycznych urządzeń we-wy, czyni możliwymi równoległe przesłania do różnych kanałów, przy czym liczba kolejek przesłań jest taka sama jak liczba kanałów.

3.1. Realizacja poziomów pamięci hierarchicznej

Poziom pamięci jest spójnym obszarem bloków odpowiadających fizycznym blokom w pamięciach masowych. Innymi słowy jest to zbiór bloków w pamięci operacyjnej lub kanałów i stacji pamięci masowych. Związek między fizycznymi jednostkami



Rys. 3. Poglądowy schemat organizacji systemu zarządzania danymi
TDZ - tablica definicji zbiorów, TDP - tablica definicji poziomów pamięci



Rys. 4. Poziomy realizacji przesłań
WE1, WE2 - wejścia dla programów użytkowych

pamięci a poziomami pamięci hierarchicznej określa tablica TDP definicji poziomów, przyporządkowująca każdemu poziomowi jego definicję zawierającą m.in. liczbę kolejnych kanałów, liczbę stacji w kanale, parametry stacji pamięci masowej, typ pamięci i algorytm numerowania bloków. Bloki informacji w fizycznych pamięciach są numerowane zgodnie z przyjętym algorytmem, przyporządkowującym każdemu blokowi numer w poziomie. Przyjęto dwa algorytmy numerowania. Jeden z nich określony jest przez wyrażenie:

$$NBP = NB + b \cdot (NP + p\,(NC + c \cdot (NS + NK \cdot s)))$$

gdzie:

	NBP	– nr bloku w poziomie pamięci,
RZECZYWISTY ADRES BLOKU	NB	– nr bloku na ścieżce,
	NP	– nr ścieżki w cylindrze,
	NC	– nr cylindra w stacji,
	NS	– nr stacji w kanale,
	NK	– nr kanału w zbiorze kanałów poziomu,
DEFINICJA POZIOMU	b	– liczba bloków w ścieżce,
	p	– liczba ścieżek w cylindrze,
	c	– liczba cylindrów w stacji,
	s	– liczba stacji w kanale.

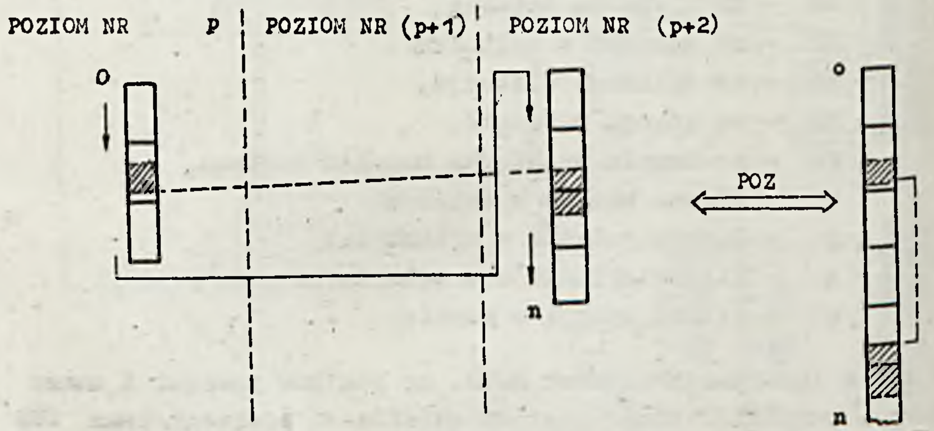
Adres informacji zawiera m.in. nr poziomu pamięci i numer bloku w poziomie. Numer poziomu określa za pośrednictwem TDP definicję poziomu. Wyznaczenie adresu rzeczywistego bloku dokonuje się na podstawie numeru bloku w poziomie, algorytmu numerowania i definicji poziomu.

Program realizacji poziomu przekształcając parametry przesłania na parametry rzeczywiste zastępuje przesłanie między programem użytkowym a poziomem pamięci hierarchicznej sekwencją przesłań między programem a fizycznymi pamięciami, a do ich realizacji wykorzystuje rozkazy we-wy programu Executive.

3.2. Zarządzanie zbiorami danych

Zbiór jest ciągiem rekordów informacji zapisanym w spójnym obszarze bloków odpowiadających blokom w poziomach pamięci. Określa go numer zbioru i długość, wyrażona liczbą zajmowanych bloków.

Zbiór dzieli się na podzbiory o numerach 0 ÷ 3 według intensywności wykorzystywania informacji. Nie wszystkie podzbiory muszą istnieć. Numer podzbioru określa jednocześnie numer poziomu pamięci, na którym jest umieszczany. Jeżeli sekwencja rekordów w zbiorze jest dzielona między poziomy pamięci, a więc występuje w różnych podzbiorach, to wszystkie jej części są powiązane w łańcuch według rosnących czasów dostępu (rys.5).



Rys. 5. Przykładowa struktura zbioru; faktyczna i widziana przez użytkownika

Wszystkie istniejące w systemie zbiory są rejestrowane w tablicy definicji zbiorów (TDZ), która określa wielkość i strukturę zbioru oraz podaje przyporządkowanie między adresem informacji w zbiorze a numerem poziomu i adresem informacji w poziomie.

Adres rekordu informacji zawiera m.in. numer zbioru, nr bloku w zbiorze i adres początku rekordu w bloku. Numer zbioru wyznacza położenie jego definicji TDZ. Nr bloku i adres

początku rekordu informacji w bloku pozwalają wyznaczyć numer poziomu i adres informacji w poziomie. Program POZ przekształcając parametry przesłania według powyższego algorytmu na parametry dotyczące poziomów zastępuje przesłanie między programem a zbiorem sekwencją przesłań między programem a poziomami.

Przedstawiona powyżej koncepcja pamięci hierarchicznej dla przetwarzanych w systemie danych jest realizowana programowo i nie stawia specjalnych wymagań odnośnie sprzętu. Optymalizuje ponadto system ze względu na koszty i czas reakcji. Ma to istotne znaczenie ze względu na występujący w wyniku zwiększających się potrzeb wzrost ilości przetwarzanej informacji.

4. ZAKOŃCZENIE

Modularne podejście do projektowania oraz niezależna od sprzętu koncepcja pamięci hierarchicznej mogą znaleźć zastosowanie także w innych systemach informatycznych.

TELSPIS - СИСТЕМА НАКАПЛИВАНИЯ И ПОИСКА ИНФОРМАЦИИ О ТЕЛЕФОННЫХ АБОНЕНТАХ

Резюме

В настоящей разработке описывается система накапливания и поиска информации" TELSPIS ", разрабатываемая Центром технической информации и обработки данных в ОЛПиТ в Варшаве, в рамках системы " TELSIT ". Обсуждается проблема модульности системы. Излагается концепция иерархической памяти и управления данными в этой памяти.

TELSPIS - A STORAGE AND RETRIEVAL SYSTEM FOR INFORMATION ABOUT TELEPHONE SUBSCRIBERS

Summary

This report shows a storage and retrieval system called "TELSPIS" realized by the Centre of Technical Informatics and Data Processing in OLPiT, Warsaw, as a part of the "TELSIT" system.

Modularity of the system is discussed.

The idea of the hierarchical storage and data management in such a storage is shown.

ALGORYTMY, Zeszyt Specjalny, 1974, 267-277

681.322.06:
061.3/438/"1973"

SYSTEM INFORMOWANIA KIEROWNICTWA RESORTU

Elżbieta I. WYSMUŁEK
Instytut Energetyki

Pracę złożono 10.I.1974

Omówiono wielodostępny system konwersacyjny KONWERS, pełniący zadanie zautomatyzowanej służby informacyjnej dla kierownictwa resortu, umożliwiający wizualną prezentację informacji ułatwiających podjęcie optymalnej decyzji. Podano opis ogólny systemu, omówiono sprzęt, zasady współpracy, konserwację i zasady pracy użytkownika z systemem KONWERS.

Spis treści

1. OGÓLNY OPIS SYSTEMU KONWERS

1.1. Wstęp

System informowania kierownictwa resortu KONWERS jest wielodostępnym systemem konwersacyjnym. Istotą systemu jest to, że w zależności od przebiegu przetwarzania informacji przez system liczący użytkownik może zmieniać program swego postępowania. Instrukcje dla systemu KONWERS podawane są przez użytkownika bezpośrednio z konsoli, a wykonywane przez system w sposób interpretacyjny. W celu uruchomienia systemu użytkownik podaje identyfikującą go informację (p. 2.2). Z chwilą, gdy KONWERS nada sygnał gotowości do współpracy, użytkownik rozpoczyna podawanie kolejnych instrukcji. Znak zakończenia pisania instrukcji spowoduje, że system rozpocznie jej wykonywanie. Przewiduje się, że z systemu KONWERS, w ciągu ustalonego okresu łączności, będzie mogło korzystać kilka komórek organizacyjnych, z których każda będzie miała przydzieloną konsolę. Nad współpracą komórek organizacyjnych z systemem KONWERS czuwa użytkownik, posiadający specjalne uprawnienia, zwany dalej menażerem.

1.2. Systemowy zbiór informacji

Używane w opisie systemu KONWERS pojęcie informacji pierwotnej rozumiemy jako informacje wprowadzone do pamięci systemu. Informacje te mogą pochodzić bezpośrednio z pierwszego źródła powstawania informacji jak również mogą przejść pracochłonny i wieloszczeblowy proces przetwarzania, przy czym, zarówno procedura przetwarzania jak i środki techniczne użyte do przetwarzania nie są istotne. W przypadku współdziałania z systemem KONWERS systemów API - informacje wtórne w tych systemach będą dla KONWERS-u informacjami pierwotnymi.

System KONWERS będzie przekazywał użytkownikowi informacje w takiej postaci, w jakiej zostały one wprowadzone do pamięci systemu. W związku z tym informacje te muszą być przetwarzane w innych systemach. Przewiduje się, że w pierwszym okresie

funkcjonowania systemu KONWERS przetwarzanie informacji i przygotowywanie konkretnych pytań odbywać się będzie poza maszyną cyfrową. Długość tego okresu zależeć będzie od uruchomienia modułowych systemów API i budowy resortowego banku informacji. Informacje pierwotne dla systemu KONWERS stanowią odpowiedzi na pewną liczbę pytań. Liczba pytań może być zmieniana w zależności od potrzeb użytkowników i pojemności pamięci pomocniczej maszyny cyfrowej. Pytania otrzymują kolejne numery od 1 do 999 (maksymalna liczba pytań w obecnej konfiguracji systemu). Ustalono, że każda odpowiedź na pytanie składać się będzie z 4 wariantów, każdy wariant z 3 porcji, każda porcja z 1040 znaków (maksymalnie). Zakładamy, że pierwsza porcja 3 wariantu (siódma kolejna) zawiera najbardziej aktualną odpowiedź na pytanie.

Całkowita lub częściowa aktualizacja danych odbywa się w czasie seansów menażera (p. 2.3) dwoma sposobami: przez wczytywanie informacji z czytnika kart lub z monitora ekranowego, albo przez odczytywanie ich z pamięci pomocniczej, gdzie zostały umieszczone za pomocą modułowych systemów API.

1.3. Konfiguracja sprzętu liczącego dla systemu KONWERS

Przewidujemy zainstalowanie w gabinetach członków kierownictwa resortu monitorów ekranowych z klawiaturą. Monitory te będą połączone z maszyną cyfrową CDC 3170 wyposażoną w:

a) pamięć operacyjną o standardowej wielkości (64 K słów 24-bitowych, czas dostępu 1,75 µs),

b) pamięć pomocniczą w postaci:
   b′) dysków (o pojemności 2 M słów każdy, średni czas dostępu 165 ms)
   b″) taśm magnetycznych (o pojemności 8 M słów każda)

c) czytnik kart perforowanych (1200 kart/min)

d) drukarkę wierszową (1300 linii/min)

Wymagana pojemność pamięci pomocniczej zależeć będzie od objętości zbioru danych, a zatem od potrzeb użytkowników. Przewiduje się, że maszyna wraz z drukarką i czytnikiem obsługiwana będzie przez specjalnie przeszkolony personel ośrodka obliczeniowego, natomiast ekrany będą do bezpośredniej dyspozycji członków kierownictwa resortu. Do jednego z ekranów musi mieć dostęp menażer systemu KONWERS. Początkowo mają być zainstalowane trzy monitory ekranowe. KONWERS dopuszcza obecnie 10 stanowisk monitorowych, ale liczbę tę można zwiększyć.

Ze względu na przeznaczenie systemu KONWERS i odpowiedzialność za udzielane przezeń informacje wszystkie dialogi przeprowadzane z systemem są rejestrowane na taśmie magnetycznej. Na specjalne żądanie otrzymuje się z zapisu wyciąg obejmujący dialogi przeprowadzone przez określonego użytkownika w określonym dniu. Po upływie ustalonego czasu (2-3 doby) zarejestrowane dialogi mogą ulec zatarciu, jeśli nikt nie zgłosił zastrzeżeń, albo jeśli nikt nie prosił o kopie.

2. WSPÓŁPRACA Z SYSTEMEM KONWERS

2.1. Instrukcje

Instrukcje systemu KONWERS podzielone są na dwie grupy: menażerskie i użytkowe.

Menażerowi wolno podawać instrukcje z obu grup, natomiast użytkownikowi instrukcje menażerskie są niedostępne. Do instrukcji tych należą: LISTA, PODPIS, TAJNE, JAWNE, UTAJNIAM, UJAWNIAM, ILOŚĆ, T. PYTAŃ, T. ODPOWIEDZI, ZMIANA, DOZWÓL, SAPI, RAPORT, CZAS, KONIEC. Instrukcjami użytkowymi są: PORCJA, KOLEJNA, WSTECZNA, POCZĄTEK, SKACZ, COFAJ, MERITUM, POWTÓRZ, TEKST, SŁOWNIK, PRZERWA, ILOŚĆ, DOŁĄCZ, DRUKUJ, KONIEC. System KONWERS dopuszcza możliwość skrócenia nazwy do pierwszych czterech znaków. Instrukcje wolno podawać tylko po otrzymaniu od systemu specjalnego komunikatu (p. 2.4.4, p. 9).

2.2. Podpis kodowy

Podpis kodowy, czyli informacja identyfikująca użytkownika, stanowi ciąg czterech znaków wybranych spośród liter i cyfr i zaczyna się literą.

Użytkownik zgłasza menażerowi swój, tajny dla innych użytkowników, podpis kodowy oraz numer monitora, przy którym będzie pracował.

Menażer przekazuje te informacje na listę systemową, a użytkownik, w konwersacji wstępnej z systemem KONWERS (p. 2.4.1) zobowiązany jest podać je jeszcze raz.

2.3. Praca konserwatora systemu KONWERS

Menażer ma możliwość współpracy z systemem w czterech seansach, dla których przyjęto następujące nazwy: konwersacja wstępna, seans pierwotny, seans wtórny i seans użytkowy.

- Konwersacja wstępna

W tym seansie menażer przedstawia się systemowi KONWERS; następnie, jeśli trzeba, wczytuje z czytnika kart pełny zbiór danych i wreszcie określa w jakich seansach chce jeszcze pracować.

- Seans pierwotny

Menażer podaje tu tylko jedną instrukcję menażerską (LISTA), która spowoduje utworzenie w pamięci systemu tablicy użytkowników uprawnionych do współpracy z systemem KONWERS. Następnie może przejść do podawania instrukcji użytkowych bądź zakończyć seans instrukcją KONIEC.

- Seans wtórny

Seans rozpoczyna instrukcja LISTA, której parametrem jest podpis kodowy menażera. Następnie menażer podaje inne instrukcje menażerskie w zależności od potrzeb, a potem ewentualnie

instrukcje użytkowe. Instrukcja KONIEC, zamykająca seans wtórny, powoduje m.in. przesłanie utworzonych przez menażera tablic do pamięci pomocniczej.

• Pozostałe instrukcje menażerskie

PODPIS – zmienia podpis kodowy menażera

TAJNE / JAWNE – utajnia (ujawnia) odpowiedzi na określone pytania dla jednej lub kilku komórek organizacyjnych

UTAJNIAM / UJAWNIAM – utajnia (ujawnia) odpowiedzi na określone pytania dla jednego lub kilku użytkowników

ILOŚĆ – informuje zarówno menażera jak i użytkownika jaką liczbą pytań dysponuje aktualnie system KONWERS. Menażer może tę liczbę zmniejszyć, a zatem może utajnić pytania o najwyższych numerach dla wszystkich komórek organizacyjnych i dla wszystkich użytkowników

T. PYTAŃ – wykonuje jedną z wymienionych czynności: ujawnia odpowiedzi na pytania utajnione instrukcją ILOŚĆ, aktualizuje teksty pytań, ujawnia i aktualizuje, ujawnia i rozszerza zbiór tekstów pytań, ujawnia, aktualizuje i rozszerza ten zbiór

T.ODPOWIEDŹ – rozszerza zbiór odpowiedzi na pytania lub aktualizuje cały ten zbiór z ewentualnym jego zwiększeniem

ZMIANA – aktualizuje wybrane porcje odpowiedzi (zmienia treść porcji, kasuje porcję lub dopisuje nową)

DOZWÓL – dopisuje do listy użytkowników uprawnionych do korzystania z systemu KONWERS tylu nowych użytkowników, na ilu pozwala miejsce w odpowiedniej tablicy systemowej

SAPI – dołącza do systemu KONWERS systemy automatycznego przetwarzania informacji

RAPORT – sporządza dokumentację pracy z systemem KONWERS określonego użytkownika w określonym dniu

CZAS – sporządza bilans czasu wykorzystanego przez danego użytkownika

KONIEC – kończy seanse pracy z systemem KONWERS

2.4. Praca użytkownika z systemem KONWERS

2.4.1. Konwersacja wstępna

Po rozpoznaniu użytkownika system KONWERS sprawdza, czy może podjąć z nim pracę, tzn. czy posiada w swej pamięci odpowiednie zbiory danych i czy w tym samym czasie nie pracuje menażer. Jeśli te warunki są spełnione, wówczas KONWERS prosi użytkownika o dokładną identyfikację, tzn. o jego podpis kodowy i numer monitora. Po podaniu tych informacji w sposób prawidłowy użytkownik może rozpocząć właściwy seans użytkowy, czyli podawanie kolejnych instrukcji użytkowych.

2.4.2. Seans użytkowy

W tym seansie użytkownik może otrzymać wszystkie informacje jakimi dysponuje dla niego system KONWERS. W tym celu podaje wybrane lub wszystkie instrukcje użytkowe.

2.4.3. Instrukcje użytkowe

PORCJA – rozpoczyna seans użytkowy, powoduje wyświetlenie na ekranie porcji odpowiedzi, której numer określają parametry instrukcji

KOLEJNA – wyświetla następną porcję odpowiedzi

WSTECZNA – wyświetla poprzednią porcję odpowiedzi

POCZĄTEK – wyświetla pierwszą porcję z danego wariantu

SKACZ – wyświetla pierwszą porcję następnego wariantu

COFAJ – wyświetla pierwszą porcję poprzedniego wariantu

MERITUM – wyświetla pierwszą porcję trzeciego wariantu

POWTÓRZ – powtarza ostatnio wyświetloną odpowiedź

TEKST – wyświetla tekst pytania, na które były udzielane odpowiedzi

SŁOWNIK – wyświetla kolejne teksty pytań, poczynając i kończąc na żądanym przez użytkownika

ILOŚĆ – informuje iloma pytaniami dysponuje KONWERS

PRZERWA – przerywa na chwilę konwersację

DRUKUJ - informuje jakie systemy API są podłączone do systemu KONWERS

DOŁĄCZ - wyświetla aktualne wyniki żądanego systemu API

KONIEC - kończy seans użytkowy

2.4.4. Teksty standardowe i diagnostyka błędów w konwersacji użytkownika z systemem KONWERS

1) SYSTEM KONWERS PYTA KIM JESTEŚ - pierwszy komunikat od systemu. Odpowiedzią winno być słowo UŻYTKOWNIK lub tylko U
2) ODPOWIEDŹ NIEZNANA SYSTEMOWI - sygnał przy nieprawidłowej odpowiedzi użytkownika w konwersacji wstępnej
3) SYSTEM KONWERS ZAJĘTY - POCZEKAJ - informacja o tym, że aktualnie pracuje menażer
4) BRAK DANYCH - ZWOLNIJ SYSTEM - w pamięci systemu KONWERS nie ma niezbędnych danych
5) PROSZĘ ZŁOŻYĆ PODPIS
6) PODAJ NUMER SWEGO MONITORA
7) NIEPRAWIDŁOWY PODPIS - system KONWERS pozwala użytkownikowi na czterokrotne poprawienie pomyłki w podpisie. Piąta omyłka eliminuje użytkownika z pracy z systemem (p.p. 18)
8) PODAJ TERAZ INSTRUKCJĘ PORCJA - system KONWERS przypomina użytkownikowi, że tą instrukcją musi zacząć seans użytkowy
9) PODAJ NASTĘPNĄ INSTRUKCJĘ - informacja, że system KONWERS jest gotowy do przyjęcia i realizacji kolejnej instrukcji użytkownika
10) NIEPRAWIDŁOWE POLECENIE - błędnie podana nazwa instrukcji
11) NIEPRAWIDŁOWA WARTOŚĆ - błędnie podany parametr instrukcji
12) KONWERS NIE PRZYJMIE ZLECENIA - zleceniem była instrukcja menażerska
13) INFORMACJA NIEDOSTĘPNA - pytanie, na które użytkownik oczekuje odpowiedzi zostało utajnione

14) BRAK DANYCH W BANKU SYSTEMU - luka w zbiorze odpowiedzi
15) PYTANIE NIE ISTNIEJE - żądanego numeru pytania nie ma w zbiorze pytań
16) KONWERS ZAWIERA ... PYTAŃ - informacja o liczbie pytań jaką dysponuje KONWERS
17) SYSTEM NIE DOŁĄCZONY DO KONWERS - żądany przez użytkownika system API nie jest jeszcze podłączony do systemu KONWERS
18) MUSISZ SKOŃCZYĆ PRACĘ - usunięcie przez system KONWERS użytkownika nieupoważnionego do współpracy
19) ZWOLNIŁEŚ MONITOR - DZIĘKUJEMY - prawidłowe zakończenie seansu użytkowego

3. ZAKOŃCZENIE

Zadanie systemu KONWERS polega na wizualnym udostępnieniu, w odpowiednio krótkim czasie, żądanych zbiorów danych na monitorach ekranowych zainstalowanych w wybranych komórkach organizacyjnych.

Wiele szczegółów nie zostało jeszcze dopracowanych ze względu na brak wyposażenia technicznego i dokumentacji zarówno sprzętu jak i firmowego oprogramowania.

System KONWERS zapewnia poufność i jest w miarę łatwy w obsłudze, jest też dostatecznie elastyczny pozwalając na dość szeroką wymianę informacji zdezaktualizowanej na świeższą, na rozszerzenie listy pytań i odpowiedzi oraz zbioru użytkowników.

System KONWERS spełnia zadanie zautomatyzowanej służby informacyjnej dla kierownictwa resortu od strony organizacji wizualnej prezentacji zamawianych informacji ułatwiających podjęcie optymalnej decyzji.

СИСТЕМА С РАЗДЕЛЕНИЕМ ВРЕМЕНИ ИНФОРМИРОВАНИЯ РУКОВОДСТВА ВЕДОМСТВА

Резюме

Система KONWERS представляет собой систему экранного изображения приготовленных порций информации. Для приведения системы в действие пользователь передает идентифицирующую его информацию. В тот момент, когда KONWERS транслирует сигнал готовности к взаимодействию, пользователь начинает передачу прикладных инструкций. В памяти системы находится не более 12 тысяч порций информации, причем каждая содержит не более 1040 знаков. Порции объединяются по три в так называемые варианты, а варианты по четыре - в большие агрегации называемые ответами на вопросы. Пользователь системы KONWERS формулируя вопрос, указывает номер ответа, номер варианта и номер порции. Если окажется, что данный пользователь уполномочен к получению указанной им порции информации, эта информация будет продемонстрирована на экранном мониторе. В определенной единице времени системой KONWERS может пользоваться 14 пользователей из 10 организационных единиц, в том числе для каждой будет предназначен определенный монитор. Кроме пользователей, системой KONWERS пользуется также "Менажер", т.е. лицо уполномоченное конструировать, обновлять систему и осуществлять за ней уход. Менажер выполняет свою задачу в менажерских сеансах, передавая инструкции, недоступные другим пользователям системы KONWERS . Система KONWERS исполняет роль автоматизированной информационной службы для руководства ведомства в отношении организации визуального изображения заказываемой информации, облегчающей принятие оптимального решения.

A MULTIACCESS SYSTEM OF INFORMING MINISTRY MANAGEMENT STAFF

Summary

The KONWERS system is a multiaccess display system for displaying prearranged portions of information. To initiate the system, the user should key up his identification code. As soon as KONWERS conveys the signal announcing its readiness to operation, the user can start to key up his successive appropriate instructions. The memory of the system contains up to 12 thousand portions of information, each having capacity not exceeding 1040 characters. The portions are grouped in threes in the so-called variants, and the variants themselves in fours in greater sets called the answers to the questions. Upon formulating a question, the user indicates answer number, variant number and portion number. If it is revealed that a given user is authorized to receive the requested portion of information, it will be displayed. In a given time unit, KONWERS can be accessed by 14 users from 10 departments, to each of which a terminal is assigned. Additionally KONWERS is used by the "Manager", i.e. a specialist authorized to install, maintain and update the system. The Manager works during so-called manager sessions and issues commands not available to other users. KONWERS meets the requirements of the automated information service for the ministry managerial staff with regard to organization of visual presentation of information wanted for facilitating the optimal decision making.

Lista uczestników sympozjum

1. W. Babczenko	Okręgowe Laboratorium Poczty i Telekomunikacji, Warszawa
2. W. Bajurski	Instytut Łączności, Warszawa
3. A. Barczak	Akademia Sztabu Generalnego, Warszawa
4. R. Bednarz	Instytut Badań Jądrowych Cyfronet, Warszawa
5. T. Berkan	Wydział Elektroniki Politechniki Warszawskiej, Instytut Maszyn Matematycznych, Warszawa
6. J. Białasiewicz	Przemysłowy Instytut Automatyki i Pomiarów, Warszawa
7. P. Bielkowicz	Instytut Maszyn Matematycznych, Warszawa
8. J. Bromirski	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
9. W. Bytnerowicz	Zakład Doświadczalny Oprogramowania przy Instytucie Maszyn Matematycznych, Warszawa
10. J. Chamski	Instytut Łączności, Warszawa
11. R. Chełstowski	Instytut Automatyzacji Systemów Zarządzania Wojskowej Akademii Technicznej, Warszawa
12. J. Chmurzyński	Wydział Cybernetyki Wojskowej Akademii Technicznej, Warszawa
13. S. Choromański	Instytut Maszyn Matematycznych, Warszawa
14. S. Chrobot	Wydział Cybernetyki Wojskowej Akademii Technicznej, Warszawa
15. T. Czachórski	Zakład Systemów Automatyzacji Kompleksowej PAN, Gliwice
16. J. Czajkowski	Krajowe Biuro Informatyki, Warszawa
17. E. Demczyszyn	Instytut Maszyn Matematycznych - Oddział Śląski, Katowice
18. A. Dernałowicz	Zakład Doświadczalny Minikomputerów przy Instytucie Maszyn Matematycznych, Warszawa
19. J. Dudziak	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
20. T. Englert	Instytut Maszyn Matematycznych, Warszawa
21. R. Faber	Politechnika Warszawska, Warszawa

22. Z. Fryźlewicz	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
23. A. Gecow	Instytut Badań Jądrowych Cyfronet, Warszawa
24. J. Grzywacz	Instytut Automatyzacji Systemów Zarządzania Wojskowej Akademii Technicznej, Warszawa
25. A. Hildebrandt	Instytut Łączności, Warszawa
26. E. Hudyma	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
27. Z. Huzar	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
28. E. Hübner-Biegalska	Instytut Maszyn Matematycznych, Warszawa
29. B. Janicka	Instytut Automatyki Przemysłowej Politechniki Warszawskiej, Warszawa
30. L. Jung	Wydział Cybernetyki Wojskowej Akademii Technicznej, Warszawa
31. J. Karczewski	Centrum Obliczeniowe Polskiej Akademii Nauk, Warszawa
32. S. Karkowski	Instytut Techniczny Wojsk Lotniczych Zespół III, Warszawa
33. J. Kasprzyk	Instytut Maszyn Matematycznych, Warszawa
34. K. Koleśnik	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
35. W. Komorowski	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
36. E. Kosmulska	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
37. Z. Kosowski	Zakład Doświadczalny Oprogramowania przy Instytucie Maszyn Matematycznych, Warszawa
38. K. Kowalski	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
39. J. Kozłowski	Instytut Automatyzacji Systemów Zarządzania Wojskowej Akademii Technicznej, Warszawa
40. R. Krasnodębski	Instytut Ochrony Środowiska, Oddział we Wrocławiu

41.	P. Kremienowski	Ośrodek Badawczo-Rozwojowy Maszyn Cyfrowych "ELWRO", Wrocław
42.	Z. Kujda	Instytut Maszyn Matematycznych, Warszawa
43.	J. Kuśnierz	Instytut Dowodzenia Akademii Sztabu Generalnego, Warszawa
44.	T. Kwiatkowski	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
45.	M. Łącka	Instytut Maszyn Matematycznych, Warszawa
46.	S. Machnik	Huta Lenina, Kraków
47.	W. Madej	Instytut Łączności, Warszawa
48.	W. Mardal	Instytut Maszyn Matematycznych, Warszawa
49.	B. Marońska	Centrum Obliczeniowe Polskiej Akademii Nauk, Warszawa
50.	J. Maroński	Centrum Obliczeniowe Polskiej Akademii Nauk, Warszawa
51.	A. Michalski	Centrum Obliczeniowe Polskiej Akademii Nauk, Warszawa
52.	Z. Michalski	Instytut Maszyn Matematycznych, Warszawa
53.	P. Misiurewicz	Instytut Automatyki Politechniki Warszawskiej, Warszawa
54.	J. Muszyński	Ośrodek Badawczo-Rozwojowy Maszyn Cyfrowych "ELWRO", Wrocław
55.	H. Mysior	Instytut Maszyn Matematycznych, Warszawa
56.	J. Mysior	Instytut Maszyn Matematycznych, Warszawa
57.	J. Olech	Instytut Maszyn Matematycznych, Warszawa
58.	A. Olszewska	Instytut Maszyn Matematycznych, Warszawa
59.	J. Olszewski	Instytut Maszyn Matematycznych, Warszawa
60.	E. Ostaficzuk	Instytut Matematyki Politechniki Warszawskiej, Warszawa
61.	T. Pajkowska	Zakład Doświadczalny Minikomputerów przy Instytucie Maszyn Matematycznych, Warszawa
62.	A. Paprocki	Zakład Doświadczalny Oprogramowania przy Instytucie Maszyn Matematycznych, Warszawa

63.	R. Pawęska	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
64.	P. Perkowski	Instytut Maszyn Matematycznych, Warszawa
65.	J. Piela	Instytut Automatyzacji Systemów Zarządzania Wojskowej Akademii Technicznej, Warszawa
66.	A. Prokop	Instytut Techniczny Wojsk Lotniczych Zespół III, Warszawa
67.	A. Przystawska	Instytut Techniczny Wojsk Lotniczych Zespół III, Warszawa
68.	J. Rakowski	Instytut Maszyn Matematycznych - Oddział Śląski, Katowice
69.	S. Roguski	Instytut Energetyki, Warszawa
70.	A. Romatowski	Państwowa Dyspozycja Mocy, Warszawa
71.	A. Rowicki	Instytut Maszyn Matematycznych, Warszawa
72.	M. Rudny	Dział Przetwarzania Informacji, Huta Warszawa, Warszawa
73.	A. Ruszkowska	Centrum Obliczeniowe Polskiej Akademii Nauk, Warszawa
74.	K. Sacha	Instytut Automatyki Politechniki Warszawskiej, Warszawa
75.	J. Schminda	Zakład Doświadczalny Organizacji Przedsiębiorstw "ORGAM", Warszawa
76.	M. Sieliwończyk	Instytut Maszyn Matematycznych - Oddział Śląski, Katowice
77.	L. Sieniawski	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
78.	P. Sienkiewicz	Akademia Sztabu Generalnego, Warszawa
79.	W. Skurzak	Instytut Automatyzacji Systemów Zarządzania Wojskowej Akademii Technicznej, Warszawa
80.	J. Sowiński	Wojskowy Instytut Łączności, Warszawa
81.	H. Stawski	Instytut Maszyn Matematycznych, Warszawa
82.	M. Suskiewicz	Instytut Automatyzacji Systemów Zarządzania Wojskowej Akademii Technicznej, Warszawa
83.	J. Szczepkowicz	Instytut Matematyczny Uniwersytetu Wrocławskiego, Wrocław

84. M. Tudruj	Centrum Obliczeniowe Polskiej Akademii Nauk, Warszawa
85. W.M. Turski	Instytut Maszyn Matematycznych, Warszawa
86. S. Waligórski	Instytut Maszyn Matematycznych Uniwersytetu Warszawskiego, Warszawa
87. J. Wierusz	Wydział Elektroniki Politechniki Warszawskiej Instytut Maszyn Matematycznych, Warszawa
88. J. Wierzbowski	Instytut Maszyn Matematycznych, Warszawa
89. I.E. Wysmułek	Instytut Energetyki, Warszawa
90. E. Zaborowska	Instytut Maszyn Matematycznych, Warszawa
91. I. Zaremba	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
92. A. Zgrzywa	Instytut Cybernetyki Technicznej Politechniki Wrocławskiej, Wrocław
93. T. Ziarko	Zakład Systemów Automatyzacji Kompleksowej Polskiej Akademii Nauk, Gliwice
94. A. Ziemkiewicz	Zakład Doświadczalny Minikomputerów przy Instytucie Maszyn Matematycznych, Warszawa

СОДЕРЖАНИЕ

CONTENTS